Broadview

Java技术大系

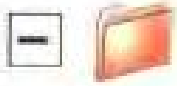
精通 Eclipse

刘亚宾 杨红 等编著

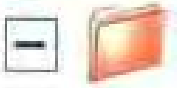


電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Broadview
www.broadview.com.cn
博文视点资讯有限公司总策划



Java 技术大系



精通 Eclipse

Eclipse是一个非常优秀的、功能强大的集成开发环境。通过集成大量的插件，Eclipse 的功能可以不断扩展，以支持各种不同的应用。Eclipse 软件可以用于管理多种开发任务，其中包括测试、性能调整及程序调试等，而且还可以集成来自多个供货商的第三方应用程序开发工具。此外，Eclipse 软件非常适合开发电子商务应用程序。

本书的目在于让读者了解 Eclipse 这一集成开发环境的体系结构及工作原理，并且通过大量的实例来指导读者如何通过利用各种插件来使 Eclipse 成为各种领域的工具，以全方位地了解 Eclipse 这一具有巨大潜力的集成开发环境。

本书包含以下内容：

- Eclipse 基础篇
- Eclipse 编程工具篇
- Eclipse 测试工具篇
- Eclipse 资源管理工具篇
- Eclipse 开发工具篇
- Eclipse 数据库应用篇
- Eclipse 高级应用篇

本书供Java程序员及各类软件开发人员阅读。

图书分类：编程语言 > Java 技术

ISBN 7-121-00384-8

9 787121 003844 >

网上订购：www.dearbook.com.cn
第二书店 · 第一服务

责任编辑：张 毅
封面设计：张子硕

本书贴有激光防伪标志，凡没有防伪标志者，属盗版图书。

ISBN 7-121-00384-8 定价：49.00 元

Java 技术大系

精通 Eclipse

刘亚宾　杨　红　等编著

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

Eclipse 是一个非常优秀的集成开发环境。本书从 Eclipse 的基础开始，介绍了 Eclipse 的体系结构、技术概况以及实际应用，由浅入深、全方位地向读者展示了 Eclipse。本书指导读者利用各种插件使 Eclipse 成为各种领域的工具，如编程工具、项目开发工具、建模工具、测试工具、版本管理工具等。本书还包含了如：Java 应用程序开发、C++应用程序开发、Web 应用程序开发、J2EE 项目开发及 Struts 框架开发等大量实例；并且介绍了 Ant，JUnit，CVS，UML 等相关技术，以及如何在 Eclipse 平台中利用这些插件进行测试、版本管理及设计等工作。全书分为七篇，共 15 章。

本书适合于 Java 程序员及各类软件开发人员阅读。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

精通 Eclipse / 刘亚宾等编著. —北京：电子工业出版社，2004.10
（Java 技术大系）
ISBN 7-121-00384-8

I. 精… II. 刘… III. 软件工具—程序设计 IV. TP311.56

中国版本图书馆 CIP 数据核字（2004）第 096829 号

责任编辑：张 毅 zhangyi@phei.com.cn
印 刷：北京大中印刷厂
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
经 销：各地新华书店
开 本：787×1092 1/16 印张：32.75 字数：794 千字
印 次：2005 年 3 月第 2 次印刷
印 数：3000 册 定价：49.00 元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：（010）68279077。质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

前　言

Eclipse 是一个非常优秀的集成开发环境，IBM 已经花了多年的时间开发基于 Java 的公开源代码软件 Eclipse，并在其中投资了 4000 万美元。这个在 IBM 支持下的开放源代码项目经过一个时期的发展完善，已经为广大 Java 开发者所熟悉。Eclipse 的出现，为 Java 开发者提供了免费使用强大的 Java IDE 的机会，通过集成大量的插件，Eclipse 的功能可以不断扩展，以支持各种不同的应用。

Eclipse 软件可以用于管理多种开发任务，包括测试、性能调整及程序调试等，而且还可以集成来自多个供货商的第三方应用程序开发工具。此外，Eclipse 软件非常适合开发电子商务应用程序，例如 IBM 公司自己的面向 Web 服务的 WebSphere 应用程序开发工具就是建立在 Eclipse 基础之上的。

除此之外，Eclipse 平台是一个成熟的、精心设计的及可扩展的体系结构。Eclipse 的价值还在于它为创建可扩展的集成开发环境提供了一个开放源码平台。这个平台允许任何人构建与环境和其他工具无缝集成的工具，而工具与 Eclipse 无缝集成的关键便是插件。

Eclipse 3.0 版本已于 2004 年 6 月 25 日发布，但是与 Eclipse 3.0 配套的插件还在陆续发布中。为了使读者在最稳定的环境下进行开发，本书将主要介绍目前使用最广泛的 2.1 版本。

本书从什么是 Eclipse 等基础知识讲起，详细介绍了 Eclipse 平台的体系结构、基本操作以及 Eclipse 各种插件的应用。本书的目的是让读者了解 Eclipse 这一集成开发环境的体系结构及工作原理，并且通过大量实例来指导读者利用各种插件来使 Eclipse 成为各种领域的工具，如编程工具、应用开发工具、建模工具、测试工具、版本管理工具等，以全方位地了解 Eclipse 这一具有无限潜力的集成开发环境。本书由浅入深，分为 Eclipse 基础、Eclipse 编程工具、Eclipse 测试工具、Eclipse 资源管理工具、Eclipse 开发工具、Eclipse 数据库应用、Eclipse 高级应用等七篇共 15 章内容。因此，本书适合于所有想了解并掌握 Eclipse 的软件开发人员。

• Eclipse 基础篇

第 1 章　Eclipse 简介

本章向读者介绍 Eclipse 的概念、主要组成、主要特点，以及下载安装与安装环境的配置等内容，在第 1 章的最后还介绍了 Eclipse 的开发环境和 Eclipse 3.0 里新增加的功能。

第 2 章　Eclipse 平台

这是本书的基础，本章将介绍 Eclipse 平台的技术概况，包括它的体系结构、各组成部分（包括运行时核心、工作台、工作区等）的工作原理，让读者对 Eclipse 平台的整体有一个充分的了解，这对以后的 Eclipse 平台操作会有帮助。另外，本章还着重介绍了对工作台的基本操作，这是使用 Eclipse 平台的基础。

• Eclipse 编程工具篇

第 3 章　利用 JDT 编写 Java 程序

本章向读者介绍的内容有：JDT 插件简介和调试、使用 JDT 进行 Java 程序的编程、调试运行，以及 JDT 的一些高级用法。在本章的最后，举了一个编写 Java 程序的实例。

第 4 章　利用 CDT 编写 C/C++程序

由于目前国内使用 C++语言的编程人员很多，Eclipse 平台也同样提供了支持 C++语言编程环境的插件 CDT。本章将向读者介绍有关 CDT 插件的功能和使用，以满足熟悉 C++语言的软件开发人员的使用需要。

• Eclipse 测试工具篇

第 5 章　Eclipse 与 JUnit

JUnit 是一个开放源代码的 Java 测试框架，用于编写和运行可重复的测试。本章除了向读者介绍有关 JUnit 知识外，还将介绍 Eclipse 如何利用本身集成的 JUnit 插件来进行单元测试。

• Eclipse 资源管理工具篇

第 6 章　Eclipse 环境下 Ant 插件的应用

本章向读者介绍有关 Ant 方面的知识，并讲解使用 Ant 插件在 Eclipse 平台上创建 Ant 构建部署文件来进行项目管理，以及与 JUnit 结合进行单元测试等。

第 7 章　使用 Eclipse 协同 CVS 进行版本管理

版本控制就是在软件开发过程中，对软件源代码文件的修改进行记录、保存的过程。版本控制对于团队开发是重要的，对于个人开发者也是项目复杂化时不可缺少的工具。在本章中，向读者介绍什么是 CVS，如何配置 CVS 服务器，Eclipse 和 CVS 的连接，以及如何使用 CVS 插件来协同 Eclipse 进行版本管理。

• Eclipse 开发工具篇

第 8 章　Web 应用程序开发

本章向读者介绍 Tomcat 服务器及其下载与安装，介绍如何利用 Tomcat 服务器插件在 Eclipse 平台上配置 Tomcat 服务器，通过一个实例来指导读者创建一个 Tomcat 项目，以及如何在 Tomcat 项目中进行 Servlet 和 JSP 的开发。

第 9 章　使用 Eclipse 进行 J2EE 项目开发

本章将向读者介绍 JBoss 服务器及其下载与安装，JBoss 是一种遵从 J2EE 规范的、开放源代码的、100%纯 Java 的 EJB 服务器，并且可以和 Tomcat 服务器完美结合。本章还介绍了 Eclipse 里的一个重要插件——Lomboz 的下载与配置。一步步地指导读者创建 J2EE 项目、利用 Lomboz 编写和调试 JSP 文件，以及利用 Lomboz 来简化 EJB 的开发。

• Eclipse 数据库应用篇

第 10 章　数据库的构筑与连接

本章向读者介绍有关数据库和数据库连接方面的知识，并通过实际构建过程介绍如何使用 JDBC 进行数据库的连接，并通过一个 MySQL 数据库的实例来指导读者进行实际操作。另外，本章还向读者介绍了 Tanghan 插件的安装和使用。Tanghan 插件主要是为 Eclipse 开发的，用来通过标准的 JDBC 接口访问数据库。在本章中，还通过生成一个

实体 EJB 的实例，向读者展示了 EJB 与数据库连接过程的实现。

• Eclipse 高级应用篇

第 11 章　Eclipse 环境下的 Struts 框架应用

Struts 是一种当今比较流行的、基于 MVC 设计模式的、用来构建 Java Web 应用程序的开源框架。在本章中，首先向读者介绍的内容有：Struts 框架简介、MVC 模式结构、Struts 的下载和安装，并通过一个简单的 Struts 应用程序实例介绍了 Struts 的结构与组成、Struts 的关键组件、Struts 标签库等方面的知识以及 Struts 的工作流程。在了解了 Struts 后，再向读者介绍如何在 Eclipse 平台上使用专门用于开发 Struts 应用程序的插件 Easy Struts 来开发 Web 应用程序，并通过实例详细讲解。

第 12 章　将 Eclipse 作为建模工具

本章将向读者介绍 Eclipse 作为 UML 建模工具的使用情况。内容包括 EclipseUML 插件的功能介绍、该插件的安装和使用 EclipseUML 插件来绘制 UML 图。

第 13 章　使用 Eclipse 进行重构

Java 开发环境的重构能力是 Eclipse 提供的最有用的特性之一。本章将向读者介绍什么是重构、重构的意义，以及如何使用 Eclipse 进行重构。

第 14 章　Eclipse 插件开发

Eclipse 最激动人心的地方就是它的插件体系结构。在本章中，将向读者介绍如何利用 Eclipse 平台提供的插件开发环境（PDE）来开发自己需要的插件。

第 15 章　Eclipse 的界面 API

本章主要介绍了 SWT 以及使用 Eclipse 制作 SWT 应用程序的方法。另外，还介绍了在 SWT 应用程序中使用 UI 流之外的其他流的方法，使用 AsyncExec 方法进行简单的调整的方法等。在最后一节中，还简单介绍了 JFace 有关方面的知识。

本书由大连高信电子先端技术有限公司组织编写，参加本书写作的人员有：刘亚宾、杨红、刘传毅、张晋宁、隋鹏和范明瑶。

同时，在本书的写作过程中，得到了大连理工大学信息学院副院长王秀坤教授、大连民族学院计算机系王德高副教授的大力支持和协助，他们提出的大量的参考性意见使本书增色不少，我们在此表示衷心的感谢。

当然，限于作者水平，加之时间仓促，书中不足之处难免，敬请广大读者批评指正。

编　者

目　录

第一第　Eclipse 基础

第 1 章　Eclipse 简介 …… 3

1.1　何谓 Eclipse …… 3
1.2　Eclipse 的主要组成 …… 4
1.3　Eclipse 的主要特点 …… 6
1.4　Eclipse 的安装 …… 7
1.4.1　JDK 的下载和安装 …… 7
1.4.2　JDK 环境变量的设定 …… 9
1.4.3　Eclipse 的下载安装 …… 10
1.5　Eclipse 的开发环境 …… 13
1.5.1　编辑窗口 …… 13
1.5.2　观察窗口 …… 14
1.5.3　透视图 …… 15
1.6　Eclipse 3.0 新功能及改进 …… 16
1.6.1　平台和 SWT 的新增功能 …… 16
1.6.2　文本的新增功能 …… 20
1.6.3　JDT 的新增功能 …… 22
1.6.4　搜索，比较，CVS，Ant 的新增功能 …… 30
1.6.5　PDE 的新增功能 …… 32
本章小结 …… 35

第 2 章　Eclipse 平台 …… 36

2.1　Eclipse 平台技术概述 …… 36
2.1.1　平台的体系结构 …… 36
2.1.2　平台运行时 …… 38
2.1.3　工作区 …… 38
2.1.4　工作台和用户界面工具箱 …… 40
2.1.5　小组支持 …… 45
2.1.6　帮助支持 …… 46
2.2　Eclipse 工作台 …… 48
2.2.1　工作台概述 …… 48
2.2.2　透视图 …… 49

2.2.3 视图 …… 50
2.2.4 编辑器 …… 50
2.2.5 工具栏 …… 52
2.2.6 第一个项目 …… 52
2.2.7 标记 …… 55
2.2.8 书签及书签视图 …… 55
2.2.9 任务及任务视图 …… 57
2.2.10 搜索及搜索视图 …… 58
2.2.11 比较 …… 61
2.2.12 本地历史记录 …… 64
2.2.13 外部工具 …… 65
2.2.14 更新管理器 …… 66
2.3 资源管理 …… 70
2.3.1 资源与工作台 …… 70
2.3.2 对资源进行操作 …… 72
2.3.3 浏览和查找资源 …… 75
2.3.4 导入/导出资源 …… 79
2.3.5 插件开发中的资源 …… 83
2.4 定制工作台 …… 86
2.4.1 定制工具栏 …… 86
2.4.2 定制键绑定 …… 87
2.4.3 更改字体和颜色 …… 91
2.4.4 更改选项卡的位置 …… 93
2.4.5 控制单击和双击行为 …… 94
2.4.6 定制透视图 …… 94
本章小结 …… 96

第二第 Eclipse 编程工具

第 3 章 利用 JDT 编写 Java 程序 …… 99
3.1 JDT 的基础操作 …… 99
3.1.1 验证 JRE 安装和类路径变量 …… 99
3.1.2 创建 Java 项目 …… 101
3.1.3 新建 Java 类 …… 103
3.2 Java 编辑器 …… 104
3.2.1 打开 Java 编辑器 …… 105
3.2.2 使用内容辅助功能编程 …… 106

3.2.3 快速修正功能 …… 108
3.2.4 管理导入语句 …… 110
3.2.5 创建和查看 Javadoc 文档 …… 112
3.2.6 标识代码指示符 …… 114
3.2.7 格式化 Java 代码 …… 116
3.2.8 使用模板编程 …… 117
3.2.9 使用本地历史记录 …… 120
3.3 JDT 的相关视图 …… 121
3.3.1 “包资源管理器”视图 …… 121
3.3.2 “大纲”视图 …… 124
3.3.3 “层次结构”视图 …… 125
3.3.4 “Java 浏览”透视图 …… 126
3.4 运行与调试 …… 127
3.4.1 运行 …… 127
3.4.2 JDT 调试器 …… 131
3.4.3 断点 …… 132
3.4.4 本地调试 …… 136
3.4.5 远程调试 …… 138
3.4.6 对表达式和代码段求值 …… 139
3.5 JDT 搜索 …… 144
3.5.1 从工作台执行 Java 搜索 …… 144
3.5.2 使用弹出菜单执行 Java 搜索 …… 146
3.6 重构支持 …… 147
3.6.1 Java 元素重命名 …… 147
3.6.2 Java 元素移动 …… 149
3.6.3 Java 元素复制 …… 151
3.7 Java 构建器 …… 151
3.7.1 构建 Java 程序 …… 152
3.7.2 构建类路径 …… 154
3.7.3 使用 JRE …… 159
本章小结 …… 162
第 4 章 利用 CDT 编写 C/C++程序 …… 163
4.1 CDT 插件介绍 …… 163
4.2 安装 MinGW …… 163
4.3 CDT 的下载和安装 …… 167
4.4 创建 C/C++工程 …… 168
4.4.1 新建一个 C++工程 …… 168

4.4.2 新建.cpp 文件 …… 169
4.4.3 新建 makefile 文件 …… 169
4.5 CDT IDE 功能 …… 170
4.5.1 自定义 C/C++编辑器 …… 170
4.5.2 代码模版 …… 172
4.5.3 代码辅助功能 …… 176
4.5.4 生成本地历史记录 …… 179
4.6 调试 C++工程 …… 179
本章小结 …… 181

第三篇 Eclipse 测试工具

第 5 章 Eclipse 与 JUnit …… 185
5.1 软件测试基础 …… 185
5.1.1 软件测试的目的 …… 185
5.1.2 软件测试的原则 …… 186
5.1.3 软件测试的对象 …… 186
5.1.4 软件测试的种类 …… 187
5.1.5 软件测试的过程 …… 187
5.1.6 软件测试的基本方法 …… 188
5.2 JUnit 简介 …… 189
5.2.1 为什么使用 JUnit …… 189
5.2.2 JUnit API 介绍 …… 191
5.3 在 Eclipse 中设置 JUnit …… 191
5.4 用 Eclipse 和 JUnit 进行单元测试 …… 193
5.4.1 单项测试 …… 193
5.4.2 多项测试 …… 199
5.4.3 多个测试类测试 …… 201
5.5 数据库程序的单元测试 …… 202
本章小结 …… 206

篇四篇 Eclipse 资源管理工具

第 6 章 Eclipse 环境下 Ant 插件的应用 …… 209
6.1 Ant 简介 …… 209
6.2 Ant 安装和配置 …… 210
6.2.1 安装 Ant …… 210

6.2.2 配置 Ant 环境变量 …… 210
6.2.3 Ant 的依赖库 …… 211
6.3 运行 Ant …… 212
6.4 build.xml 语法详解 …… 213
6.4.1 项目（project） …… 214
6.4.2 目标（target） …… 214
6.4.3 任务（task） …… 216
6.4.4 属性（property） …… 216
6.4.5 过滤器（token filters） …… 218
6.4.6 路径指定元素 …… 218
6.4.7 命令行变量 …… 219
6.5 使用 Ant 插件 …… 221
6.5.1 创建 Ant 构建文件 …… 221
6.5.2 编辑 Ant 构建文件 …… 222
6.5.3 运行 Ant 构建文件 …… 225
6.5.4 保存和重用 Ant 选项 …… 227
6.6 用 build.xml 构建 Ant 部署文件 …… 229
6.7 Ant 和 JUnit 结合进行单元测试 …… 234
本章小结 …… 236

第 7 章 使用 Eclipse 协同 CVS 进行版本管理 …… 237

7.1 什么是 CVS …… 237
7.2 CVS 服务器的安装 …… 237
7.3 在 Linux 平台上的配置 CVS 服务器 …… 239
7.4 在 Windows NT 平台上的配置 CVS 服务器 …… 242
7.5 应用 CVS 系统协助开发的一个简单的流程 …… 242
7.6 CVS 的几个基本概念 …… 244
7.7 Eclipse 和 CVS 的连接 …… 245
7.8 启动一个新项目 …… 248
7.9 应用 Eclipse 从 CVS 资源库检出代码 …… 251
7.10 在开发过程中 Eclipse 协同 CVS 开发的基本操作 …… 254
本章小结 …… 260

第五第 Eclipse 开发工具

第 8 章 Web 应用程序开发 …… 263

8.1 Web 服务器 …… 263

8.1.1 Web 服务器简介 …… 263
8.1.2 在 Linux 下安装 Apache 和 Tomcat …… 264
8.1.3 在 Windows 下安装 Tomcat …… 268
8.2 Tomcat 的配置介绍 …… 269
8.2.1 Tomcat 目录结构 …… 269
8.2.2 server.xml 配置文件介绍 …… 269
8.3 Eclipse 与 Tomcat 的连接 …… 271
8.3.1 Tomcat 插件下载安装 …… 271
8.3.2 Tomcat 在 Eclipse 上的配置 …… 271
8.4 在 Eclipse 平台上开发 Web 应用程序 …… 274
8.4.1 新建 Tomcat 项目 …… 274
8.4.2 创建 Tomcat 项目中的 Servlet …… 276
8.4.3 创建 Tomcat 项目中的部署描述符 web.xml …… 277
8.4.4 创建 Tomcat 项目中的 JSP …… 278
本章小结 …… 281
第 9 章 使用 Eclipse 进行 J2EE 项目开发 …… 282
9.1 J2EE Web 应用服务器 …… 282
9.1.1 介绍几种常用的 J2EE 服务器 …… 282
9.1.2 JBoss 下载与安装 …… 284
9.2 Lomboz 插件的下载与安装 …… 285
9.2.1 Lomboz 插件的介绍 …… 285
9.2.2 Lomboz 插件配置 …… 285
9.3 创建 J2EE 项目 …… 287
9.4 利用 Lomboz 编写和调试 JSP 文件 …… 291
9.5 Eclipse 结合 Lomboz 开发 EJB …… 293
9.5.1 XDoclet 介绍 …… 294
9.5.2 生成 EJB …… 294
9.5.3 生成 EJB 接口类 …… 297
9.5.4 EJB 部署 …… 299
9.5.5 生成客户端 EJB …… 300
9.5.6 运行 JBoss 服务器 …… 301
本章小结 …… 302

第六篇 Eclipse 数据库应用

第 10 章 数据库的构筑与连接 …… 305
10.1 数据库的构筑 …… 305

10.1.1 MySQL 数据库简介 305
10.1.2 MySQL 数据库安装 306
10.1.3 MySQL 数据库基本操作指令 306
10.2 数据库连接 309
10.2.1 JDBC 对象 309
10.2.2 JDBC 驱动类型 310
10.2.3 JDBC 的使用过程 311
10.2.4 一个简单的数据库查询的例子 312
10.3 Tanghan 插件 313
10.3.1 Tanghan 插件简介 313
10.3.2 Tanghan 插件下载及安装 315
10.3.3 使用 Tanghan 插件连接数据库 315
10.3.4 使用 Tanghan 插件进行数据库查询及修改操作 317
10.3.5 数据库连接测试程序 318
10.4 EJB 与数据库的连接 319
10.4.1 JBoss 的数据库设定 319
10.4.2 生成实体 Bean 320
10.4.3 实体 Bean 的部署和运行 323
本章小结 326

第七第 Eclipse 高级应用

第 11 章 Eclipse 环境下的 Struts 框架应用 329
11.1 Struts 框架简介 329
11.2 MVC 模式 329
11.2.1 MVC 模式结构 329
11.2.2 Model 2 体系结构 330
11.2.3 MVC 设计模式的小结 331
11.3 Struts 的下载和安装 331
11.4 一个简单的 Struts 程序 332
11.5 Struts 的组成与结构 340
11.5.1 Struts 应用程序组件 340
11.5.2 MVC 模式下的 Struts 344
11.5.3 Struts 的配置文件 345
11.6 Struts 标签库 349
11.6.1 HTML 标签 350
11.6.2 Logic 标签 356

11.6.3 Bean 标签 …… 359
11.7 Easy Struts 插件 …… 362
11.8 用 Easy Struts 插件开发 Struts 项目 …… 363
11.8.1 新建 Struts 工程 …… 363
11.8.2 生成 JSP 文件 …… 366
11.8.3 生成 ActionForm 类 …… 367
11.8.4 生成 Action 类 …… 369
11.8.5 生成 test.properties 文件 …… 372
11.8.6 配置 struts-config.xml 文件 …… 373
11.8.7 生成数据库访问文件 …… 374
11.8.8 编辑文件 …… 376
11.8.9 实例运行 …… 380
11.8.10 实例小结 …… 381
本章小结 …… 383
第 12 章 将 Eclipse 作为建模工具 …… 384
12.1 何谓 EclipseUML 插件 …… 384
12.2 EclipseUML 插件的安装 …… 386
12.3 利用 EclipseUML 插件绘制 UML 图的基础 …… 387
12.4 利用 EclipseUML 插件绘制 UML 图的工程实例 …… 391
12.4.1 在系统分析阶段中描述用况 …… 391
12.4.2 在概要设计阶段中抽出概念模型 …… 392
12.4.3 在概要设计阶段中根据概念模型绘制顺序图 …… 395
12.4.4 在详细设计阶段中根据所制成的顺序图补充类图 …… 396
12.4.5 在“编程过程”中整理 Package …… 398
12.4.6 在“编程过程”中分割组件，整理部署 …… 400
12.4.7 其他图简介 …… 401
本章小结 …… 402
第 13 章 使用 Eclipse 进行重构 …… 404
13.1 什么是重构 …… 404
13.2 为什么要进行重构 …… 404
13.3 使用 Eclipse 进行重构的实例 …… 405
13.3.1 抽取方法 …… 407
13.3.2 抽取常量 …… 410
13.3.3 移动 …… 412
13.3.4 更改方法特征符 …… 415
13.3.5 内联 …… 417

13.3.6 抽取局部变量 …… 419
13.3.7 重构撤销与重构重做 …… 421
13.3.8 抽取接口 …… 421
13.3.9 重命名 …… 424
13.3.10 拉出与下推 …… 425
本章小结 …… 431

第 14 章 Eclipse 插件开发 …… 432

14.1 Eclipse 平台体系结构 …… 432
14.2 插件简介 …… 433
14.3 开发一个简单的视图插件 HelloWorld …… 434
14.3.1 创建插件 …… 434
14.3.2 安装和运行插件 …… 440
14.4 插件开发环境简介 …… 443
14.4.1 主机和运行时 …… 444
14.4.2 外部插件与工作区插件 …… 444
14.4.3 配置 PDE …… 445
14.4.4 PDE 提供的视图 …… 446
14.5 使用 PDE 创建插件 …… 447
14.5.1 创建插件 …… 448
14.5.2 插件清单编辑器 …… 450
14.6 运行插件 …… 456
14.6.1 选择要运行的插件 …… 457
14.6.2 对运行时进行跟踪 …… 458
14.6.3 在调试方式下运行 …… 460
14.6.4 运行插件 …… 460
本章小节 …… 462

第 15 章 Eclipse 的界面 API …… 463

15.1 SWT 简介 …… 463
15.1.1 SWT 的特征 …… 463
15.1.2 SWT 应用程序结构 …… 465
15.1.3 SWT 提供的组件 …… 465
15.1.4 布局 …… 468
15.1.5 SWT 生命周期 …… 473
15.2 SWT 应用程序开发 …… 474
15.2.1 创建 SWT 项目 …… 474
15.2.2 创建 SWT 应用程序 …… 477

15.2.3 Widget 的事件注册侦听器处理 …… 485
15.3 线程问题 …… 486
15.3.1 本机事件调度 …… 486
15.3.2 工具箱用户界面线程 …… 487
15.3.3 SWT 用户界面线程 …… 488
15.3.4 执行非用户界面线程中的代码 …… 488
15.3.5 线程化 …… 489
15.3.6 向 UI 线程移送处理 …… 491
15.4 插件的用户界面框架（JFace） …… 493
15.4.1 查看器 …… 494
15.4.2 操作和添加项 …… 495
15.4.3 用户界面资源 …… 496
15.4.4 长时间运行的操作 …… 499
本章小结 …… 499
附录 A 部分专业术语中英文对照 …… 500
附录 B 部分资源下载地址 …… 502

第一篇

Eclipse 基础

PART 1

第 1 章 Eclipse 简介

第 2 章 Eclipse 平台

第 1 章　Eclipse 简介

2001 年 11 月 IBM 宣布捐出价值 4000 万美金的开发软件给开放源代码的 Eclipse 项目。如此受到青睐的 Eclipse 到底是什么样子呢？其引人注目之处究竟在哪里呢？本章将会对其做一介绍。

在本章中，将介绍以下内容：

- Eclipse 是一个非常优秀的集成开发环境，它旨在简化用于多操作系统软件工具的开发过程，可以用于管理多种开发任务，同时又是一个成熟的、精心设计的、可扩展的体系结构。
- Eclipse 主要由 Eclipse 项目（Eclipse Project）、Eclipse 工具项目（Eclipse Tools Project）和 Eclipse 技术项目（Eclipse Technology Project）三个项目组成。
- Eclipse 的主要特点：开放的、可扩展的 IDE，成功的底层图形界面 API，强大的插件加载功能，可以通过导入一些 CVS 相关的插件来实现版本管理的便易性，充分的帮助文件。
- 如何安装 Eclipse 2.1 版本。
- 了解 Eclipse 的最新版本 Eclipse 3.0 的新功能及改进之处。

1.1　何谓 Eclipse

Eclipse 是一个非常优秀的集成开发环境（IDE）。IBM 花了多年的时间来开发基于 Java 的公开源代码软件 Eclipse，这个在 IBM 支持下的开放源码项目经过一段时期的发展完善，已经为广大 Java 开发者所熟悉。Eclipse 是一个用于构建集成 Web 和应用程序开发工具的平台，它本身并不会提供大量的最终用户功能，而是通过插件来快速开发集成功能部件。

Eclipse 提供了公共用户界面模型来使用工具。Eclipse 软件旨在简化用于多操作系统软件工具的开发过程，它被设计成可以在多个操作系统上运行，基于 Eclipse 的软件在 Linux 和微软 Windows 系统上都可以运行，因此可以省去开发者有时要把 Windows 应用程序切换到 Linux 的操作，进而简化了整个开发过程。同时 Eclipse 还提供了与每个底层操作系统的强大集成。

Eclipse 的核心是动态发现插件的体系结构。平台负责处理基本环境的后台工作，并提供标准的用户导航模型，于是每个插件可以专注于执行少量的任务。Eclipse 的出现，不仅为 Java 开发者提供了免费使用强大的 Java IDE 的机会，而且也为 C++的开发者提供了 CDT（C 语言开发工具）。

通过集成大量的插件，Eclipse 的功能可以不断扩展，以支持各种不同的应用。Eclipse 软件可以用于管理多种开发任务，其中包括测试、性能调整以及程序调试等，而且还可以集成来自多个供货商的第三方应用程序开发工具。此外，Eclipse 软件非常适合开发电子商务应用程序，例如 IBM 公司自己的面向 Web 服务的 WebSphere 应用程序开发工具就是建立在 Eclipse 基础之上的。

除此之外，Eclipse 平台是一个成熟的、精心设计的、可扩展的体系结构。Eclipse 的价值还在于它为创建可扩展的集成开发环境提供了一个开放源码平台。这个平台允许任何人构建与环境和其他工具无缝集成的工具，而工具与 Eclipse 无缝集成的关键便是插件。

1.2 Eclipse 的主要组成

在学习如何使用 Eclipse 之前，有必要对关于 Eclipse 项目的主要组成做一些解释和说明。

Eclipse 是一个开放源代码的软件开发项目，专注于为高度集成的工具开发提供一个全功能的、具有商业品质的工业平台。它由 Eclipse 项目、Eclipse 工具项目和 Eclipse 技术项目三个项目组成，最近又新增了 Eclipse Web 工具平台项目，每一个项目由一个项目管理委员会监督，并由它的项目章程管理。每一个项目由其自身的子项目组成，并且使用 CPL 版本 1.0 许可协议。

为了更加直观地了解 Eclipse 项目的主要组成部分，即 Eclipse 项目、Eclipse 工具项目和 Eclipse 技术项目三个项目及其子项目，本书提供了截至 2003 年 3 月的主项目和子项目的构成图以供参考，如图 1-1 所示。

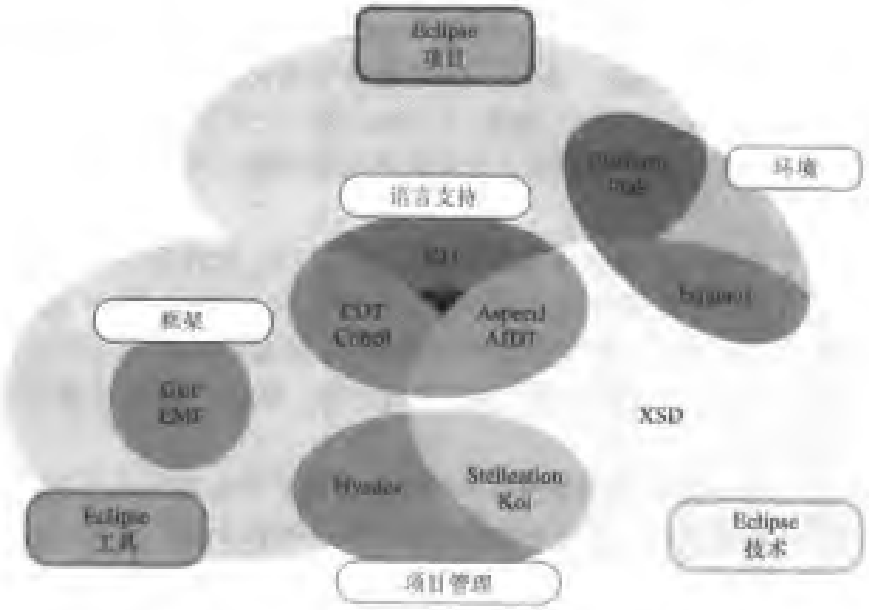


图 1-1 Eclipse.org 分解的主项目及其子项目

Eclipse 工具项目为不同的工具建造者提供了一个焦点，以保证为 Eclipse 平台创建最好的工具。Eclipse 工具项目的任务是为 Eclipse 平台培育广泛的创建工具。工具项目提供单一的联系点以调和开放源代码工具建造者，从而使得覆盖和重复最小化，并保证共享的最大化和共同组件的创建，促进不同类型工具的无缝互操作。工具项目由工具开发者委员会与工具项目的项目管理委员会提议、选择和开发的子项目组成。

Eclipse 技术项目的任务是为开放源代码开发者、研究者、学院和教育者提供新的管道，以参与将来 Eclipse 的演化。它按照研究、培育和教育三个项目流来组织：研究项目在 Eclipse 相关领域诸如编程语言、工具和开发环境方面进行探索和研究；培育项目是小型的、未正式结构化的项目，为 Eclipse 软件基础添加新的能力；教育项目聚焦于教育材料的开发、教学帮助和课件。

表 1-1 对图 1-1 做了简要说明，更加系统地介绍了 Eclipse 的项目组成。

表 1-1　Eclipse 项目组成

主项目名	子项目名	概　要
Eclipse	Eclipse 的核心项目、焦点、提供了一个具有丰富特性的开发环境，允许开发者有效地建造可以无缝集成到 Eclipse 平台中的工具	
	Platform	开发作为 Eclipse 的集成开发环境的核心部分
	Java Development Tools（JDT）	开发 Java 开发环境
	Plug-in Development Environment（PDE）	开发插件开发环境
Eclipse Tools	开发 Eclipse 所使用的工具。在 Eclipse 中，可以通过各种各样的插件来实现功能的扩展。该项目不但提供了开发功能的工具，同时兼备防止生成相同插件的功能	
	C / C++ IDE（CDT）	开发 C / C++使用的开发环境
	Graphical Editor Framework（GEF）	开发用于图片编辑的平台
	Eclipse Modeling Framework（EMF）	开发由模块生成编码的平台
	Cobol	开发 COBOL 使用的开发环境
	Hyades	开发用于测试或跟踪等质量管理方面的工具
Eclipse Technology	进行 Eclipse 的研究以及教育	
	AspectJ	关于将 Java 扩展为面向方面编程语言（AOP）的研究
	AspectJ Development Tools（AJDT）	开发 AspectJ 使用的插件
	Koi	便于多用户合作开发的结构
	Stellation	软件的组成管理
	XSD	解析 XML Schema 的 Schema Infoset Model
	Equinox	关于 Eclipsed 的构成信息，包括动态插件以及依存关系管理等

1.3 Eclipse 的主要特点

1. 开放的可扩展的 IDE

Eclipse 平台是一个开放的可扩展的 IDE。Eclipse 平台是建造模块和构造并运行集成软件开发工具的基础。Eclipse 平台允许工具建造者独立开发与其他工具无缝集成的工具，用户甚至无须去分辨一个工具功能在哪里结束、而另一个工具功能在哪里开始。

2. 成功的底层图形界面 API

从 Java 诞生至今，已经在太多的领域取得成功，然而它却很少在图形界面程序上崭露头角。究其原因，Java 语言缺省的图形界面开发包 AWT 和 SWING 实在是难辞其咎，无论速度和外观，它们都难以让人接受。如今，Eclipse 组织编写的 SWT 开发包，为 Java 程序员提供了 AWT 和 SWING 之外的一个更佳的选择。

SWT 本身仅仅是 Eclipse 组织为了开发 Eclipse 集成开发环境所编写的一组底层图形界面 API。或许是无心插柳，又或是有意为之，至今为止，SWT 无论是在性能和外观上，都超越了 Sun 公司提供的 AWT 和 SWING。目前 Eclipse IDE 已经开发到了 2.1 版本，SWT 已经十分稳定。这里指的稳定应该包含两层意思：

（1）指性能上的稳定，其中的关键是源于 SWT 的设计理念。SWT 最大化了操作系统的图形构件 API，就是说只要操作系统提供了相应图形的构件，那么 SWT 只是简单应用 JNI 技术调用它们，只有那些操作系统中不提供的构件，SWT 才自己去做一个模拟的实现。可以看出 SWT 性能上的稳定大多时候取决于相应操作系统图形构件的稳定性。

（2）指 SWT API 包中的类、方法的名称和结构已经少有改变，程序员不用担心由于 Eclipse 组织开发进度快（Eclipse IDE 每天都会发布一个 Nightly 版本），而导致自己的程序代码变化过大。从一个版本的 SWT 更新至另一版本，通常只需要简单将 SWT 包换掉就可以了。

3. 强大的插件加载功能

可以通过不断地加载插件来实现同其他制品的合作。整个 Eclipse 体系结构就像一个大拼图，可以不断地向上加插件，同时，现有插件上还可以再加插件，进而实现功能的扩展。目前，Eclipse 已经开始提供 C 语言开发的功能插件。更难能可贵的，Eclipse 是一个开放源代码的项目，任何人都可以下载 Eclipse 的源代码，并且在此基础上开发自己的功能插件。也就是说未来只要有人需要，就会有建立在 Eclipse 之上的 COBOL，Perl，Python 等语言的开发插件出现。同时可以通过开发新的插件扩展现有插件的功能，比如在现有的 Java 开发环境中加入 Tomcat 服务器插件。可以无限扩展，而且有着统一的外观、操作和系统资源管理，这也正是 Eclipse 的潜力所在。

4. 可以通过导入一些 CVS 相关的插件来实现版本管理的便易性

Eclipse 平台提供了对于直接从工作区进行团队开发操作的支持。这种支持允许开发人员并发地与几个独立的资源库以及不同版本的代码或项目进行交互。工作区中的资源允许团队支持组件处理版本和配置管理问题。当然，单个工作区也可以同时访问不同类

型的资源库。Eclipse 平台并没有提供它自己的代码管理解决方案，它总是依靠外部系统。Eclipse 平台只对一个（但也是最流行的一个）源代码管理系统提供内置支持——并发版本控制系统 CVS。

Eclipse 提供了强大的个人版本管理机制，每一次被保存的更改都可以得到恢复，而且可以精确到每一个方法的版本恢复。操作也十分方便，在任何一个能看到所要操作文件的观察窗口中，例如资源浏览窗口，选中该文件，单击右鼠标键，选择“Compare with”或“Replace with”即可。如果是恢复已经被删除的方法，则可以选择“Add from local history”，之后相应的本地历史记录就会显示出来，按照用户本人的需求找到相应的版本就可以了。强大的个人版本管理功能为程序员提供了更多的信心——只管编下去，任何不小心的错误都可以恢复到从前的状况，在 Eclipse 下开发，是有“后悔药”的！

5．充分的帮助文件

值得提出的是，在下载了 Eclipse 的多国语言翻译包之后，所有的菜单和联机帮助都已经翻译成了中文，使用起来非常方便。

1.4　Eclipse 的安装

1.4.1　JDK 的下载和安装

使用 Eclipse，必须先安装 JDK。如果已经安装了 JDK，那就可以跳过这两节，直接安装 Eclipse。如果没有安装过 JDK，可以参考以下说明。

JDK 是许多 Java 专家最初使用的开发环境。尽管许多编程人员已经使用第三方的开发工具，但 JDK 仍被当作 Java 开发的重要工具。JDK 包含了所有编写、运行 Java 程序所需要的工具：Java 基本组件、库、Java 编译器、Java 解释器、小应用程序浏览器，以及一些用于开发 Java 应用程序的程序等。其核心 Java API 是一些预定义的类库，开发人员需要用这些类来访问 Java 语言的功能。Java API 包括一些重要的语言结构以及基本图形、网络和文件 I/O。一般来说，Java API 的非 I/O 部分对于运行 Java 的所有平台是相同的，而 I/O 部分则仅在通用 Java 环境中实现。作为 JDK 实用程序，工具库中有七种主要程序：

（1）Javac——Java 编译器，将 Java 源代码转换成字节码。

（2）Java——Java 解释器，直接从类文件执行 Java 应用程序字节代码。

（3）Appletviewer——小程序浏览器，一种执行 HTML 文件上的 Java 小程序的 Java 浏览器。

（4）Javadoc——根据 Java 源码及说明语句生成 HTML 文档。

（5）Jdb——Java 调试器，可以逐行执行程序，设置断点和检查变量。

（6）Javah——产生可以调用 Java 过程的 C 过程，或建立能被 Java 程序调用的 C 过程的头文件。

（7）Javap——Java 反汇编器，显示编译类文件中的可访问功能和数据，同时显示

字节代码含义。

安装 JDK 可选择 Sun 的 JDK 或 IBM 的 JDK 都可以，应该安装 1.3 以上版本，推荐使用 1.4 以上版本，因为只有使用 1.4 以上版本的 JDK 才可以享受到新增的 HotSwap 热切换功能给调试带来的方便。下面是一个 JDK 的下载网址 http://www.monga.info/pub/Java/j2sdk-1_4_1_03-windows-i586.exe。下载完后进行安装，双击运行"j2sdk-1_4_1_03-windows-i586.exe"文件，将出现一个欢迎安装 JDK 的界面，如图 1-2 所示，单击"Next"按钮继续，按照安装提示进行安装即可。



图 1-2 JDK 安装界面

为 JDK 指定安装目录，默认是 C:\j2sdk1.4.1_03，如果你想修改安装路径，就单击"Browse"按钮，本机的安装路径为 H:\j2sdk1.4.1_03，如图 1-3 所示。安装路径选择好后，按"Next"按钮继续。

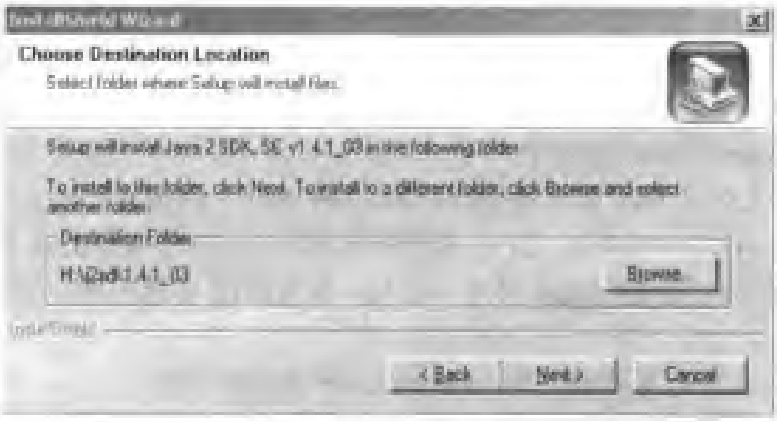


图 1-3 JDK 安装路径选择画面

接着，安装程序将要求选择要安装的组件。如图 1-4 所示，在想安装的组件前面的方框上打上一个勾。可选组件包括：

- Program Files——这是 JDK 的主体文件，必选组件。
- Native Interface Header Files——本地接口头文件，必选组件。
- Demos——一些关于 Java 的演示，可选组件。

- Java Sources——一些 Java 的示例源程序，可选组件。
- Java 2 Runtime Environment(Windows drive only)——Java 运行时间环境，可选组件，建议选中。

选择完合适的安装组件后，单击“Next”按钮。

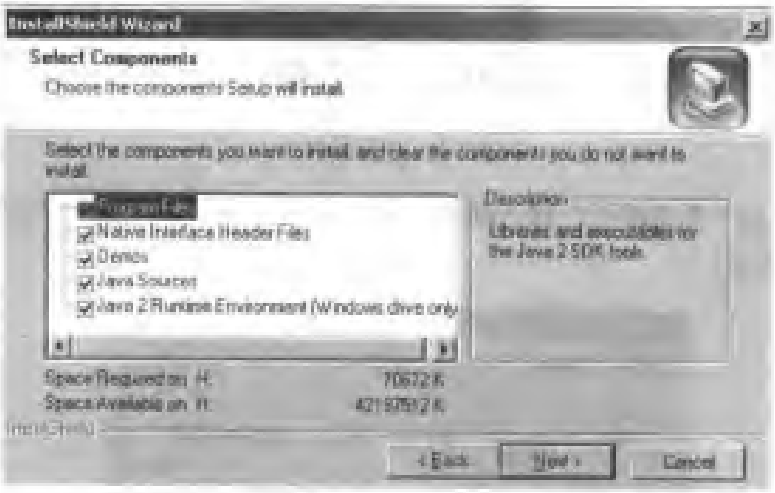


图 1-4　选择 JDK 安装组件画面

在下面的出现的画面中，安装程序将要求是否选择微软浏览器，建议选中。单击“Next”按钮安装，如图 1-5 所示。

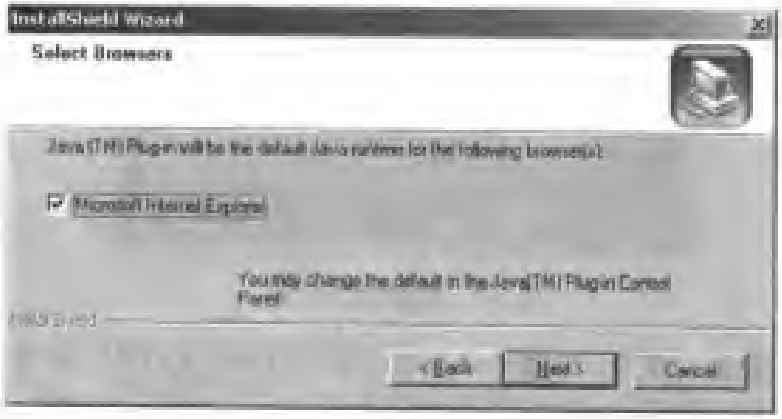


图 1-5　选择浏览器画面

最后，按“Finish”按钮，JDK 安装结束。

1.4.2　JDK 环境变量的设定

安装完 JDK（J2sdk1.4.1_03）后，需要设置计算机系统的环境变量。本机使用的是 Windows 2000 的操作系统，设置 JDK 环境变量的步骤如下：

（1）选中“我的电脑”，点鼠标右键选择“属性（R）”，如图 1-6 所示。

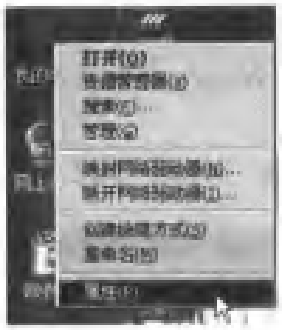


图 1-6　电脑属性

（2）在弹出的“系统特性”对话框中选择“高级”标签页，单击“环境变量（E）...”按钮，如图 1-7 所示。

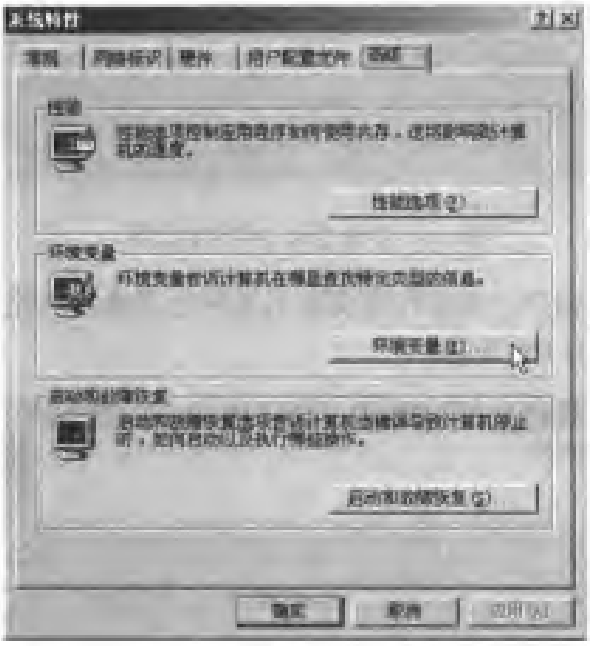


图 1-7　系统特性对话框

（3）单击“新建”按钮，新建一个环境变量“HOME_PATH”。如本机的 JDK 安装在 H:\目录下，所以 HOME_PATH 的变量值就设置为“H:\j2sdk1.4.1_03”，如图 1-8 所示。

（4）单击“确定”后再编辑 Path 变量，选中 Path 变量，单击“编辑”，在 Path 变量值后面加“;H:\j2sdk1.4.1_03\bin;”，如图 1-9 所示。

单击“确定”。至此，JDK 系统环境变量就设置完了。

1.4.3　Eclipse 的下载安装

Eclipse 是个免费的软件，到 Eclipse 的官方网站可以得到它。其中包括了开发 Java 应用的所有内容和 Eclipse 项目的所有成果。



图 1-8　新建环境变量画面



图 1-9　编辑环境变量画面

进入 Eclipse 项目的主页 http://www.eclipse.org，单击 Downloads，选择一个自己认为比较近的镜像点，进入下载页。Eclipse 目前最新的版本是 3.0 版本，发布日期是 2004 年 6 月 25 日。由于 3.0 版本的相关插件还在陆续开发中，因此在这里将下载稳定的、已被广泛应用的 2.1.2 版本。单击 2.1.2 进入其下载页，这个页面有很多下载连接，包括 Eclipse SDK 在很多平台上的版本。这里只介绍 Windows 平台下的 Eclipse，所以需要下载 Windows 平台的 Eclipse 安装文件的版本 eclipse-SDK-2.1.2-win32.zip，同时还要下载多国语言包的插件 eclipse2.1.2-SDK-win-LanguagePackFeature.zip。Eclipse 不需要安装，下载完了以后，将 eclipse-SDK-2.1.2-win32.zip 压缩文件直接解压到 H:\路径下，解压后文件夹的结构如图 1-10 所示。



图 1-10　eclipse 文件目录画面

同时将多国语言包解压后的 plugins 文件夹与 features 文件夹覆盖 eclipse 文件夹下的同名文件夹。然后，双击 eclipse 文件夹下面的 eclipse.exe 执行文件，启动 Eclipse，会闪现一个很酷的月蚀图片，如图 1-11 所示。



图 1-11　Eclipse 2.1 启动画面

之后，Eclipse 会显示它的缺省用户界面，如图 1-12 所示。

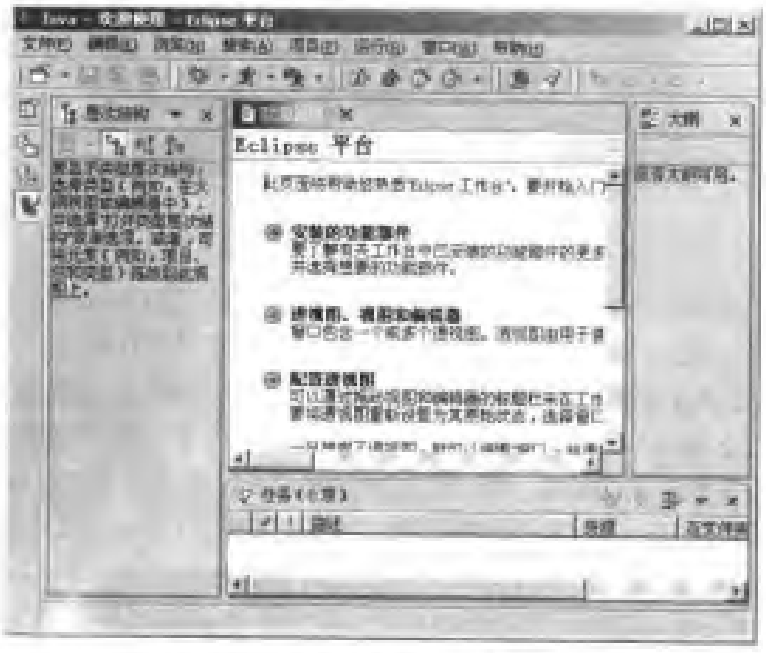


图 1-12　成功安装 Eclipse 后的初始界面

1.5　Eclipse 的开发环境

下面对 Eclipse 的开发环境作一个简单的介绍。希望读者能了解 Java 透视图、Java 包结构浏览窗口、资源透视图、资源浏览窗口等之间的区别（其实最主要的是要区别透视图和窗口），以免在进一步的学习中产生混淆。图 1-13 是它们之间的关系结构略图。

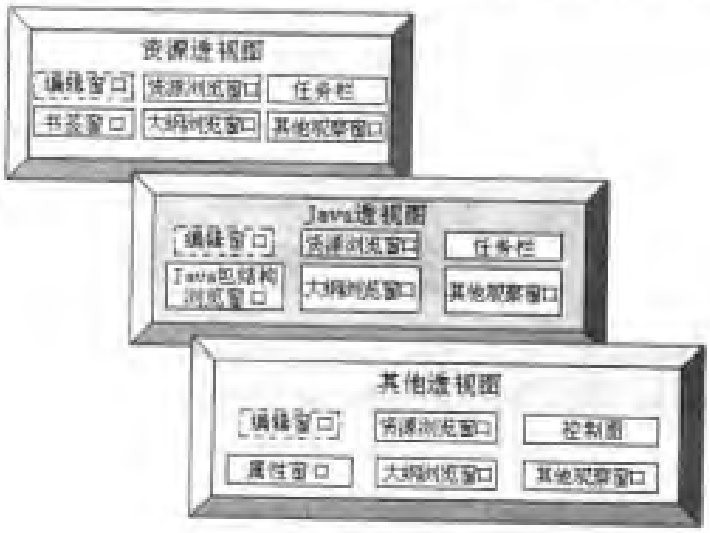


图 1-13　Eclipse 工作台的关系结构略图

在图 1-13 中，可以看出工作台（Workbench）包含多个透视图，而每个透视图又包含不同的窗口。由于每个编辑窗口有很大的共性，而且缺省情况它们都在同一区域中显示，因此我们只在每个透视图中标出一个编辑窗口，以虚线框表示。浏览窗口则各不相同，这里以实线框表示。Eclipse 的工作台如图 1-14 所示。

1.5.1　编辑窗口

所有文件的显示和编辑都包含在编辑窗口（Editor）里，缺省情况下打开的多个文件是以标签方式在同一个窗口中排列出来的，可以用拖动标签的方式将这些文件排列成各种布局。方法是拖动某一个文件的标签到编辑窗口的边框，当光标有相应的变化时再释放。

当文件被加入到项目中后，在资源浏览窗口或 Java 包浏览窗口中双击文件，Eclipse 会试图打开这个文件；其中 Eclipse 内嵌的编辑器能默认打开一些文件，如*.java，*.txt，*.class 等。如果是其他类型的文件，Eclipse 会调用操作系统相应的默认编辑器打开，如 Word 文档、PDF 文件等。同时 Eclipse 也可以像 IVJ（IBM VisualAge for Java）一样用指定的编辑器打开相应的文件。例如在 Eclipse 项目中双击 HTML 文件时，如果希望是用 Notepad 打开，而不是用系统默认的 IE 浏览器打开的话，实现的方法是打开菜单栏中的 WorkBench Preferences 对话框，之后在对话框中选择 Workbench File Editors，然后添加文件类型，如*.html，再为其指定编辑器即可。



图 1-14 Eclipse 工作台

在编辑窗口，还值得注意的是，习惯了 IVJ 的程序员在编辑 Java 程序的时候，更愿意以方法作为独立的编辑单位（即在编辑窗口中只显示单个的方法，而不是程序全部的源代码），这种方式的确是非常合理的开发方式，不仅代码的显示更加简洁，还能辅助程序员编出封装性更好的类。在 Eclipse 的工具栏上提供了一个切换按钮，可以在“显示全部代码”和“只显示所选单元”（这里的单元指的是单个方法、变量、导入的包等）之间切换。建议没有使用过 IVJ 的程序员也尝试一下在“只显示所选单元”状态下进行代码开发。

1.5.2 观察窗口

观察窗口（Observation）配合编辑窗口提供了多种的相关信息和浏览方式。常用的观察窗口有资源浏览窗口（Navigator），Java 包浏览窗口（Packages），控制台（Console），任务栏（Task）等。观察窗口是任何 IDE 的核心，用好观察窗口也就能很好地使用 IDE。Eclipse 提供了丰富的观察窗口，能真正用好这些窗口需经过一段时间的磨练。

资源浏览窗口和 Java 包浏览窗口是观察窗口核心部分。前者和 Windows 的浏览器差不多，能浏览项目文件夹中的所有文件，后者用来浏览项目中的 Java 包、包中的类、类中的变量和方法等信息。在 Java 包浏览窗口中可以通过右击鼠标，在菜单中选择“打开层次结构”来打开层次结构浏览窗口（Hierarchy），这个窗口非常实用，能非常清晰地查看类的层次结构，如图 1-15 所示。类的编译出错信息可以在任务窗口中查到，同时它也可以被称为“名符其实”的任务窗口：向其中添加新的任务描述信息，用以跟踪项目的进度。控制台则主要用来显示程序的输出信息。在调试程序的时候，还会有更丰富

的观察窗口供程序员使用，以便程序员能够更加有效地进行调试，如变量值察看窗口，断点窗口等。



图 1-15　类的层次结构图

1.5.3　透视图

一个透视图（Perspective）包括一个或多个编辑窗口和观察窗口。在开发环境的最左侧的快捷栏中的上边部分显示的就是当前所打开的透视图图标，如图 1-16 所示。透视图是 Eclipse 的最灵活的部分，可以自定义每个透视图中包含的观察窗口种类，也可以自定义一个新透视图。这些功能都被包括在“透视图”菜单中。在 Eclipse 的 Java 开发环境中提供了几种默认透视图，如资源透视图（Resource Perspective，它也是第一次启动 Eclipse 时的默认透视图），Java 透视图（Java Perspective），调试透视图（Debug Perspective），CVS 资源浏览透视图（Team Perspective）等。每一种透视图都对应不同种类的观察窗口。可以从窗口菜单栏中的“显示透视图”看到该透视图对应的观察窗口。当然，每个透视图的观察窗口都是可配置的，可以在窗口菜单栏中的“定制透视图”来进行定制。多样化的透视图不但可以帮助程序员从不同角度观察代码，也可以满足不同的编程习惯。



图 1-16　Eclipse 透视图快捷键

1.6 Eclipse 3.0 新功能及改进

2004 年 6 月 25 日 Eclipse 组织发布了 Eclipse 的最新版本 Eclipse 3.0。为了让读者对 Eclipse 3.0 有一个感性的认识，本节将对 Eclipse 3.0 的新增功能做一个详细介绍。

与以往的 Eclipse 版本相比，Eclipse 3.0 主要改进了以下五个方面：

- Eclipse 平台和 SWT
- 文本
- JDT
- Ant、搜索、比较、小组和 CVS
- PDE

下面来详细介绍一下这些改进的具体内容。

1.6.1 平台和 SWT 的新增功能

1.6.1.1 平台的改进

1. 新感观

Eclipse 3.0 在感观上有了很大地改善，界面更加精致美观，如图 1-17 所示。

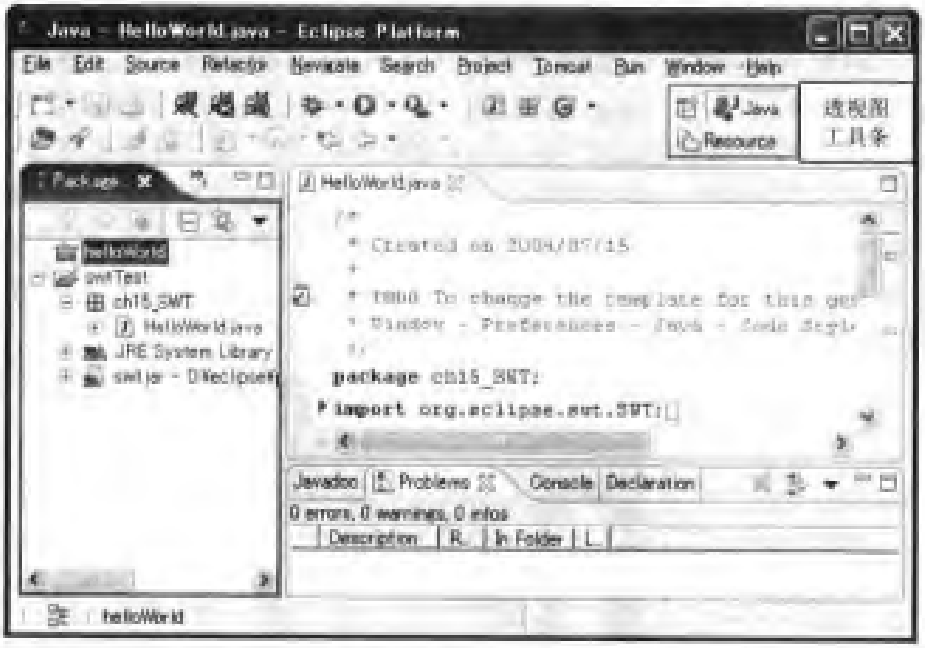


图 1-17 Eclipse 3.0 平台

主要包括以下几个方面的改善：

- 视图和编辑器上的标题栏和标签改变为柔和的弧线型。
- 视图和编辑器上的标题栏和标签中出现了最大化和还原的按钮。

- 视图上方加入折叠的按钮。
- 透视图转换/打开的工具栏发生变化。该工具栏默认被放置在窗口的右上方，用户也可以将其放置在左上方或者左侧。透视图按钮上显示文字内容，便于用户快速识别当前透视图。
- 视图拖放操作更加方便。
- 支持分离窗口（由于平台的限制，仅限于 Windows 和 Linux GTK）。
- 编辑器管理发生改变。
- 视图内部的工具栏可以被放置到视图标签的后面以便尽可能地节省空间。
- 不同标签的字体和颜色也不相同，有助于更清晰地显示活动的和被选择的视图和编辑器。

2. 快速响应的 UI

Eclipse 3.0 版本在 UI 上做出了一些改进，提高了 UI 的响应能力。新版本支持后台运行，从而改善了运行程序时需要等待的状况。

Eclipse 3.0 版本新增了以下内容：

- 进程视图
- 用于显示正在后台运行的进程状态栏
- 用于显示在后台运行的操作的对话框

新的进程视图显示了 CVS 检验的进程情况，同时还显示了后台中正在构建的某个工作区，如图 1-18 所示。后台支持多用户操作，当用户看到带有“Run In Background”按钮的进程对话框时，可以选中该按钮，这样一来，正在运行的操作将被转移到后台去运行，从而允许用户继续其他工作。该对话框还为用户显示了当前正在工作区内运行的操作的详细信息。

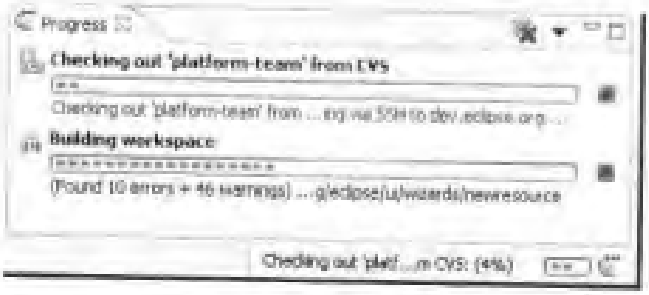


图 1-18　进程视图

3. 可缩放的 UI

用户可能因为不同的理由而使用工作台，为此工作台提供了一些附加的方法来加以控制，减少不必要的 UI 构件，以免让用户感到混乱。新版本追加了以下的功能：

- 允许过滤当前不需要的功能。
- 可有效地添加 UI 构件。
- 可根据所选择对象的类型有条件地添加菜单和工具栏（例如，只有在显示 XML 文件时，才会出现“Run”→“AntBuild...”的选项）。

4. 编辑器管理

编辑器管理新增的功能主要包括以下四点：

- 支持单一和多个编辑标签。单一的编辑标签特别适用于那些习惯打开很多文件或者习惯用键盘控制操作的用户。
- 提供编辑器锁定功能。当一次被打开的编辑器数量被限定时，一些编辑器会因为被锁定而无法关闭。当某个编辑器被锁定时，该编辑器的四周会出现一个明显的边界来指示该编辑器已被锁定。
- 添加箭头符号来指示实际打开的编辑器比显示在屏幕上的标签多出多少，如图 1-19 所示。
- 新的菜单选项及键盘快捷键绑定能更有效地帮助用户进行编辑器的管理。例如，以下是新增的菜单选项及快捷键：
 - “Close Others”——关闭除当前编辑器之外的所有编辑器。
 - “Close All”——关闭所有打开的编辑器。
 - “Ctrl+E”——调出编辑器所支持的下拉菜单，如图 1-19 所示。

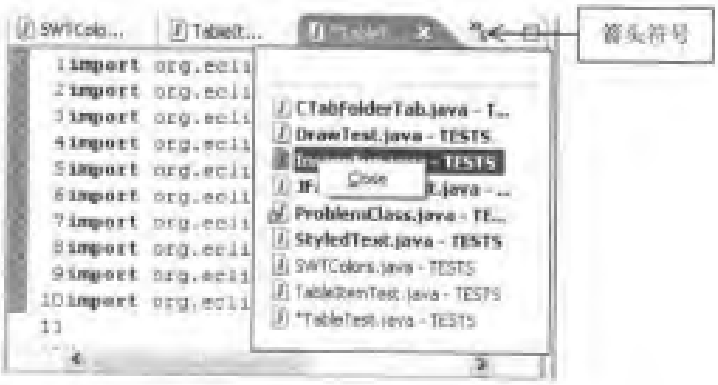


图 1-19 编辑器的下拉菜单

5. 后台工作区自动更新功能

新版本中，当本地文件系统发生改变时，工作区可以提供自动更新的服务，这样可以使用户从频繁的手动更新中解放出来。这种新功能在默认状态下是不可使用的，不过可以在工作台的首选项页面中进行设置。

6. 简化的手动构建指令

新版本将“Project”菜单下的一系列可利用的构建指令进行了简化。旧版本中，用户易混淆的“Rebuild All”和“Rebuild Project”指令，在新版本中已经被单独的“Clean...”指令所代替，如图 1-20 所示。新增的“Build Working Set”子菜单可以让用户选择要构建的项目，这个指令可以把所有的项目放到一个工作组，如果需要的话也可以构建任何不在工作组中的项目。通过菜单中的“Build Automatically”指令快速指定关闭或者打开自动构建。

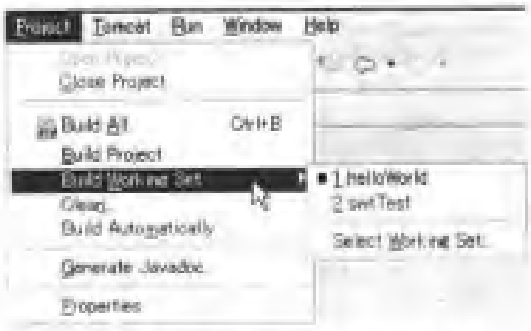


图 1-20　"Project" 菜单中的构建指令

7. Cheat Sheets

Eclipse 3.0 新增了具有多步骤的 Cheat Sheets，为用户提供向导服务。在 Cheat Sheets 中包含了一些非常实用的指南功能，例如为用户提供的创建 Java 应用程序的向导服务，如图 1-21 所示。



图 1-21　简单 Java 应用程序的 Cheat Sheets 窗口

8. RCP

Eclipse 3.0 包括了 RCP，这是 Eclipse 3.0 区别于 Eclipse 2.1 的一个显著特征。Eclipse 2.1 的平台只是作为 IDE 被设计的，而 Eclipse 3.0 则被设计为一个通用的工具集成平台。Eclipse 的很多组件并不只专门用于 IDE 中，对非 IDE 的应用程序同样有意义。新版本改善了 Eclipse 的插件体系结构、运行时和工作台的功能，可适应于构建任意的应用程序。

1.6.1.2　SWT 的改进

1. SWT 浏览器部件

在 Eclipse 3.0 中，SWT 新增了一个浏览器部件，能够显示 HTML 类型的文件，如图 1-22 所示。该部件支持以下操作系统：Windows（使用 Internet Explorer 5.0 以上版本）、Linux GTK、Linux Motif（Mozilla 1.4 GTK2、Mozilla 1.5 GTK2、Mozilla 1.6 GTK2 均支持）、Mac（Safari）以及 QNX（Voyager）。



图 1-22　用 SWT 浏览器显示 HTML 文件

2. SWT 支持复杂文字和从右向左书写的语言

为了能够实现这一功能，必须解决几个难题。例如，希伯来语和阿拉伯语都是由右向左书写的，在进行描述之前，需要依照 Bidi 算法将文本描述为 Unicode annex #9。另外一些语言，例如阿拉伯语和波斯语，它们对文字的上下文位置要求非常严格，也就是说，同一个单词位于句子中的不同位置，可能会代表不同的意思。另一个共同难题就是，如何将这些不同风格的语言绑定在一起。像泰国语之类的语言，它们在文字断点和光标导向上有一些特殊的规则。Windows 支持所有上面提到过的语言，在 GTK 中，虽然支持 Bidi 算法，但是对于一些复杂文字（比如泰国语、南亚语）的支持还不完备。

3. SWT 的可视化表格

在新版本中，表格支持 SWT.VIRTUAL 类型，这使得开发人员可以快速创建拥有大容量数据的表格。当需要添加表格数据项时，能够得到立即响应，如图 1-23 所示。

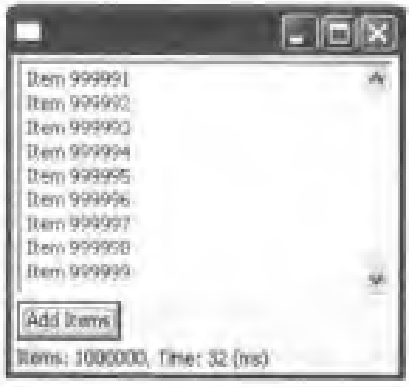


图 1-23　表格的可视化

1.6.2　文本的新增功能

1. 快速比较（Quick Diff）

在编辑器的垂直标尺栏上新增一个用色彩变化做标记的指示符，可以通过在标尺栏上弹出的上下文菜单或者按“Ctrl+Shift+Q”快捷键方式打开或者关闭快速比较功能。

在文本编辑器中增加、删除和更改的部分在标尺栏上将有颜色变化，并且当光标放置在发生变化的标尺栏上方时，会出现一个悬浮窗口，用来显示修改前的内容，如图 1-24 所示。而且，可以通过上下文菜单中的“Set QuickDiff Reference”指令将原文件内容保存下来。由于比较算法会消耗大量内存，如果当前版本与参考版本比较变动很大时，快速比较会自动暂挂，这样可以避免系统承受内存压力。

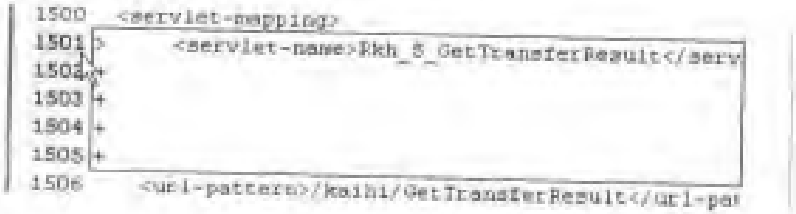


图 1-24　用于显示原内容的悬浮窗口

2. 共享注释首选项

新增的注释首选项用于配置所有基于文本编辑器的注释。除了使用波浪形下划线，注释还能在编辑器中进行高亮显示。通过在首相项页面中选择“Highlight in text”的复选框来实现。另外，“Show in vertical ruler”的复选框用来控制在垂直标尺栏上注释的可视性，如图 1-25 所示。

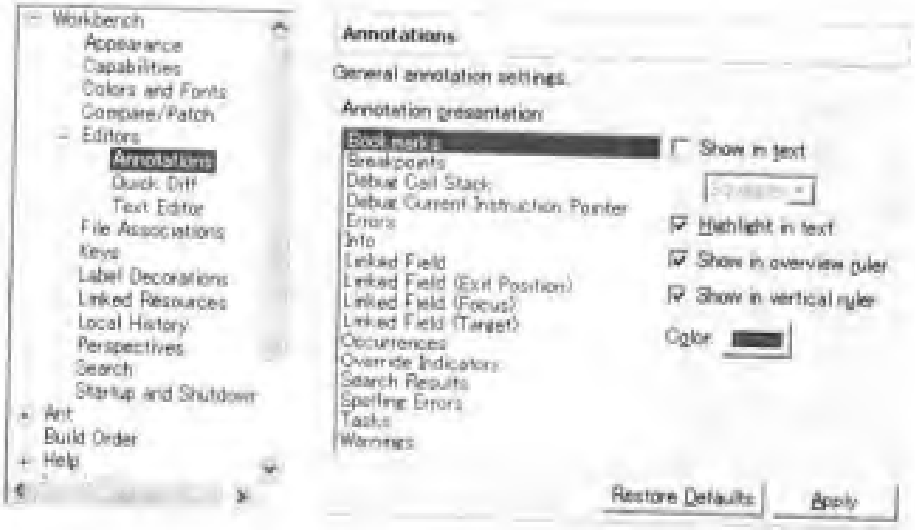


图 1-25　注释首选项页面

3. 编辑器的新增功能

所有文本编辑器都基于 Eclipse 编辑器框架，因此文本编辑器同样支持编辑器的一些新增功能，包括上下行间的移动（按“Alt+Arrow Up”或者“Alt+Arrow Down”键），行的拷贝（按“Ctrl+Alt+Arrow Up”键），字母大小写的转换（按“Ctrl+Shift+X”或者“Ctrl+Shift+Y”键）等。

1.6.3 JDT 的新增功能

1.6.3.1 Java 编辑器的新增功能

1. 语法的高亮显示

新版本中的 Java 编辑器能够依照自己的语法对源代码进行高亮显示（例如，静态字段、本地变量、静态方法调用）。在 Java 编辑器首选项页面中，选择“Syntax”选项卡，选中“Enable advanced highlighting”复选框后，新的高亮显示选项就会出现在列表中。

2. 新增的代码折叠功能

Eclipse 3.0 的 Java 编辑器新增代码折叠功能。只要将鼠标悬浮在折叠的 Java 元素上，被隐藏的代码即被显示出来，如图 1-26 所示。



图 1-26 Java 编辑器代码折叠

当前可被折叠的内容包括导入声明、注释、类型和方法体。在 Java 编辑器首选项页面中，在“Folding”选项卡中进行可折叠项目的设置。另外，JDT UI 提供一个扩展点，用来扩充可利用的可折叠项目。

3. 粘贴时自动更新导入功能

拷贝和剪切 Java 代码可以捕获到被拷贝的代码所需的导入声明的信息，新增的功能是当粘贴代码时，需要的声明也将自动导入。这个特性可以通过在 Java 编辑器首选项的“Typing”选项卡页面中，选中“Update imports on paste”复选框来实现。

4. 代码格式化的新增功能

新版本对 Java 代码格式化进行了许多改进。在代码格式化首选项页面中，允许用户选择已经存在的格式，或者定义自己的格式，或者与其他人共享格式。新增格式化功能具有很高的可配置性，超过 140 种选项，可覆盖从花括号的位置到换行边线再到 Javadoc 注释的一切内容。

另外，新版本提供了对多个文件同时进行格式化的功能，可以同时对 Java 项目、源文件夹或者包进行格式化。

5. 拼写检查功能

Java 编辑器中新增了拼写检查功能。在拼写首选项页面中设置好拼写检查项目并指定一个字典之后，在 Java 编辑器中就可以拼写检查了，并且还能获得相应的快速修正提示，如图 1-27 所示。

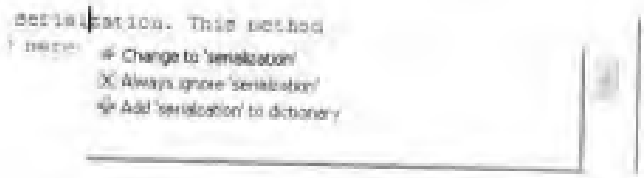


图 1-27　拼写检查及快速修正功能

6. 在文件中动态地标记事件

在 Java 编辑器中所选元素能够被动态地高亮显示。单击“Mark Occurrences”按钮（ ）或者按“Alt+Shift+O”快捷键之后，只要选中编辑器中的一个元素，则编辑器中该元素的所有调用被高亮显示，如图 1-28 所示。

```
String [] = {"1",null,null,"4"};
[0] = "0";
[1] = null;
[2] = null;
[3] = "3";
```

图 1-28　调用被高亮度显示

这种元素包括“变量”、“方法”、“常量”、“非常量域”、“本地变量”、“表达式”、“返回值”。可以通过“Windows”→“Preferences”→“Java”→“Editor”→“Mark Occurrences”来选择哪种元素使用“Mark Occurrences”，如图 1-29 所示。

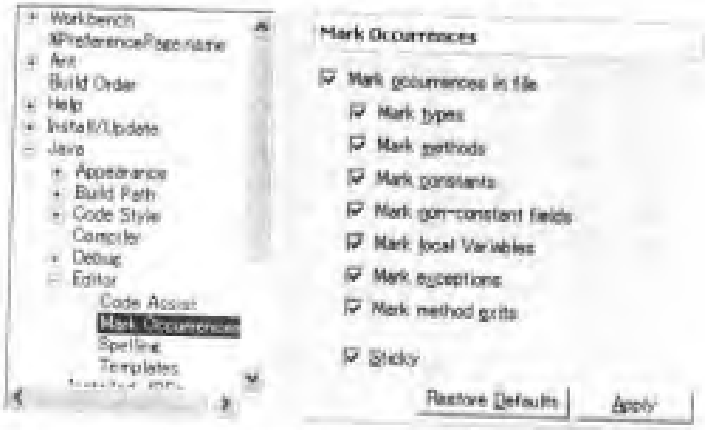


图 1-29　“Mark Occurrences”首选项页面

7. 快速修正不适当的参数

Eclipse 3.0 增加了对一些不适当参数的快速修正，如图 1-30 所示，参数“a”为不适当的参数，可以根据“修正建议”进行修正。

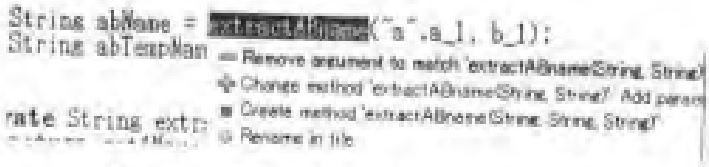


图 1-30　根据“修正建议”修正不适当参数

1.6.3.2 Java 工具的新增功能

1. 方法调用的层次视图

用户可以在 Java 编辑器中或者任何可以显示方法的视图中通过“Navigate”→“Open Call Hierarchy”打开调用层次视图，该视图用于显示此方法在哪一个类中被调用过。如图 1-31 所示。

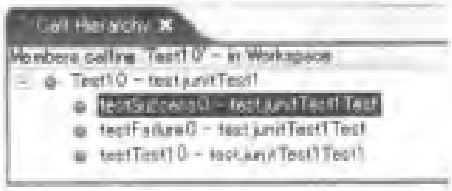


图 1-31 调用层次视图

2. Javadoc 视图

新增的 Javadoc 视图可以显示用户在 Java 编辑器中或者在 Java 视图中所选元素的 Javadoc。用户可以通过“Window”→“Show View”→“Other”→“Java”→“Javadoc”打开这个视图。如图 1-32 所示。

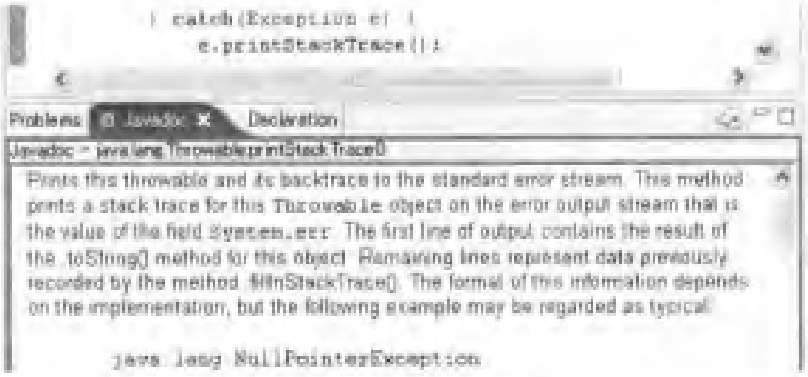


图 1-32 Javadoc 视图

3. 声明视图

新增的声明视图用来显示用户在 Java 编辑器中或者 Java 视图中所选元素的源代码。用户可以通过“Window”→“Show View”→“Other”→“Java”→“Declaration”打开这个视图。如图 1-33 所示。

4. 类型过滤器

在新版本中，可以对代码辅助和快速修复中出现的类型的集合进行过滤。用户可以同过“Window”→“Preferences”→“Java”→“Type Filters”使用这个功能。在过滤列表中出现的过滤模式将不会在“Open Type”对话框中出现，也不会在代码辅助和快速修复建议中出现。但是，这些被过滤的类型仍然会显示在包浏览器和类型层次视图中。

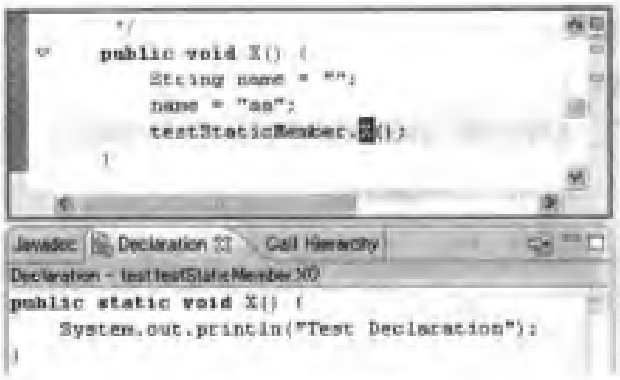


图 1-33　声明视图

5. 搜索视图的新增功能

在新的搜索视图中，用户可以选择显示搜索结果的布局，目前有两种布局可供用户选择，它们是简单布局（Flat Layout）和有层次的布局（Hierarchical Layout）。“Java 搜索”结果还可以按照项目、包、文件或者类型进行分组。用户还可以过滤“Java 搜索”结果，例如，可以将 Javadoc 注释、导入语句，以及用来访问数据的读操作和写操作的搜索结果过滤掉，如图 1-34 所示。

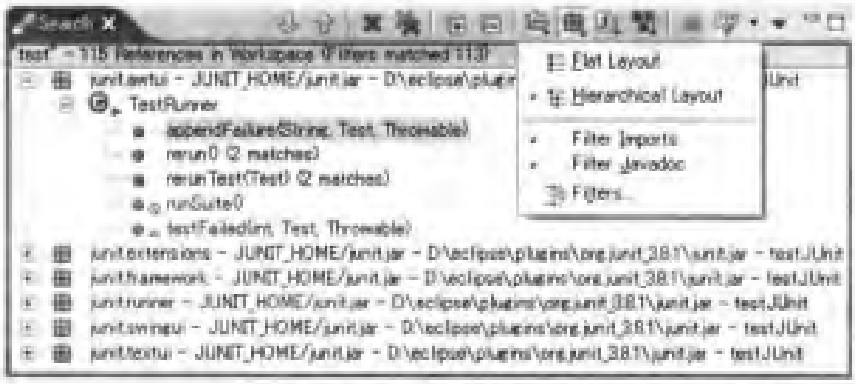


图 1-34　搜索视图

6. 对话框中的内容辅助

Eclipse 3.0 在各种 Java 对话框中都可以使用内容辅助功能。当用户将光标移至可以使用此功能的输入框的时候，输入框的前方会出现一个小灯泡样子的提示符（💡）。例如，在新建类的对话框中，输入新建类的父类名字的输入框就有此功能。

1.6.3.3　Java 重构的新增功能

1. 新增源（source）和重构（refactoring）的快捷菜单

源和重构能够通过快捷菜单来调用。在 Java 编辑器或 Java 视图中选择一个元素，然后按“Alt+Shift+S”快捷键可以打开源的快捷菜单，按“Alt+Shift+T”快捷键可以打开重构的快捷菜单。

2. Introduce Factory 重构

在 Java 编辑器中选择一个类的构造函数或它的调用，然后选择“Refactoring”→“Introduce Factory”，这时出现“Introduce Factory”对话框，如图 1-35 所示。

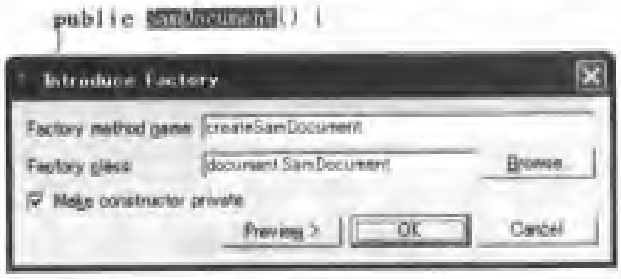


图 1-35 “Introduce Factory”对话框

3. Introduce Parameter 重构

在 Java 编辑器中选择一个表达式，然后选择“Refactoring”→“Introduce Parameter”，这时出现“Introduce Parameter”对话框，如图 1-36 所示。



图 1-36 Introduce Parameter 重构

4. Extract method 查找重复代码片断

可以通过 Extract method 重构查找重复代码片断，从而能够更好地抽取新方法。

1.6.3.4 Java 编译器的新增功能

1. 对布尔型变量的无效赋值语句的标示

在 Eclipse 3.0 中，Java 编译器可以找到并标示出程序代码中对布尔类型变量的无效赋值语句。用户可以通过“Preferences”→“Java”→“Compiler”→“Style”来设置此种问题的级别（默认的级别是忽略）。如果将其级别设置为警告（Warning），那么，在出现这种问题的时候，用户会收到警告提示，如图 1-37 所示。

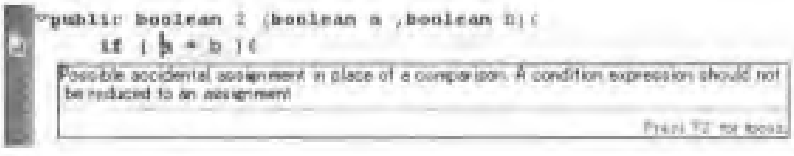


图 1-37 对布尔型变量的无效赋值语句的标示

2. 对不能被正确编译的 finally 代码块的标示

Java 编译器可以找到并标示出程序中不能被正确编译的 finally 代码块，如图 1-38 所示。用户可以通过“Preferences”→“Java”→“Compiler”→“Style”来设置此种问题的级别（默认的级别是警告）。

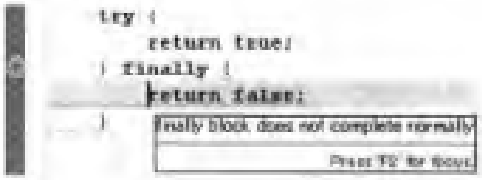


图 1-38　对不能被正确编译的 finally 代码块的标示

3. 对空代码块的标示

Java 编译器可以找到并标示出程序中空方法或空类型体，所有这些都将会被标示出来，如图 1-39 所示。用户可以通过“Preferences”→“Java”→“Compiler”→“Style”来设置此种问题的级别（默认的级别是忽略）。

4. 对代码中不必要的类型转换的标示

Java 编译器可以找到并标示出程序中不必要的强制转换和“instanceof”操作，如图 1-39 所示。用户可以通过“Preferences”→“Java”→“Compiler”→“Unused Code”来设置此种问题的级别（默认的级别是忽略）。

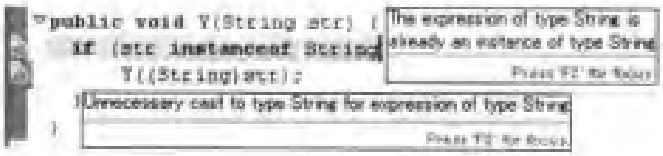


图 1-39　对代码中不必要的类型转换的标示

5. Java 编译器的快速修正功能

新版本的 Java 编译器提供了快速修正功能。如图 1-40 所示，用户可以通过以下三种方式实现此功能：

- 将光标移至代码中出错的位置，按下“Ctrl+1”快捷键。
- 单击出错代码行之前的警告标志（）。
- 通过“Edit”→“Quick Fix”使用此功能。

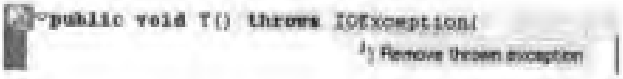


图 1-40　Java 编译器的快速修正功能

6. 对 Javadoc 注释的处理

新版本的 Java 编译器可以对 Javadoc 注释进行处理。用户可以通过“Preferences”

→“Java”→“Compiler”→“Javadoc”来设置此种问题的级别（默认的级别是忽略），如图 1-41 所示。



图 1-41　修改问题的级别

如果将级别设置为“警告”，那么，在 Java 编辑器中不正常的 Javadoc 的注释会被标示出来，如图 1-42 所示。

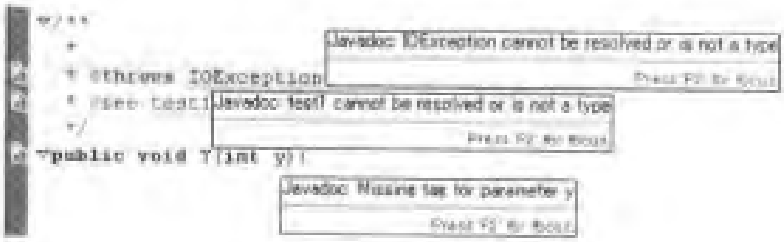


图 1-42　对不正常的 Javadoc 注释的标示

1.6.3.5　Java 调试器的新增功能

1．用于检查或显示的弹出窗口（pop-up）

新版本中的赋值弹出窗口现在可用于所有的显示和检查结果中。方法是：在想要显示结果的变量上右击鼠标，在弹出的下拉菜单中选择“display”或“inspect”，如图 1-43 所示的是在下拉菜单中选择“inspect”后的显示。

2．系统线程过滤器

在 Eclipse 3.0 中，系统线程可以通过系统线程过滤器从调试视图（Debug View）中过滤掉。在调试视图中选择下拉菜单中的“Show System Threads”选项，将会显示系统线程，如图 1-44 所示，默认状态下不选择此项。

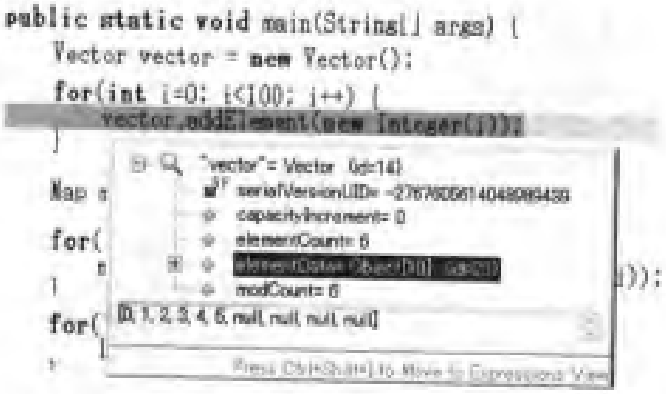


图 1-43　用于检查的弹出窗口

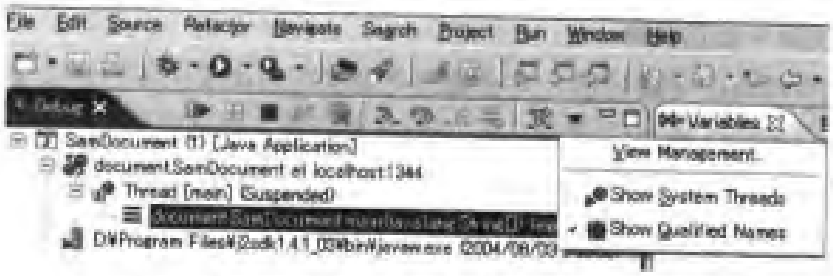


图 1-44　选择下拉菜单中的“Show System Threads”选项

3. 类预备断点

断点可能被设置在一个类的预备事件上。当这个被指定的类或接口第一次被 Java 虚拟机加载时程序将被挂起。

4. 当热码置换失败时提供重启选项

当热码置换失败时 Eclipse 3.0 会提供一个“Hot code replace failed”对话框，在该对话框中提供重启（Restart）、继续（Continue）、终止（Terminate）三个按钮。

5. 跳过断点

断点视图中新增的“Skip All Breakpoints”按钮可以跳过在工作台上的所有断点，如图 1-45 所示。



图 1-45　断点视图

6. 数组的过滤器

可以在变量视图中过滤掉值为“null”的数组元素，在变量视图的下拉菜单中选择“Show Null Array Entries”，显示值为“null”的数组元素，如图 1-46 所示。

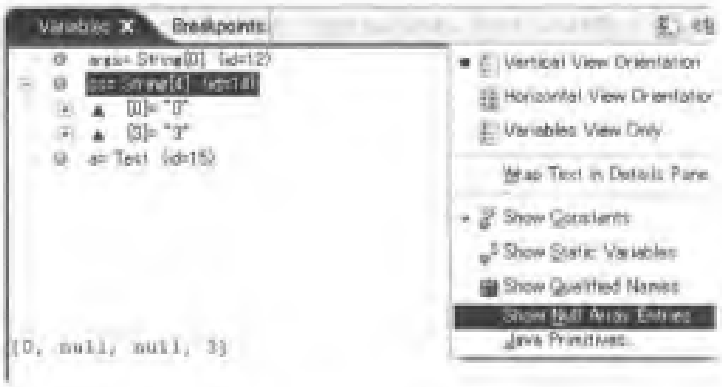


图 1-46　变量视图

7．在变量视图中设定变量

在变量视图中可修改变量的值。方法是在变量视图的详细区域中修改变量值后，使用下拉菜单中的“Assign Value”指令（或 Ctrl+S 组合键）进行保存。

1.6.4　搜索，比较，CVS，Ant 的新增功能

1.6.4.1　搜索的新增功能

1．在文件搜索中可以使用规则表达式

在搜索对话框的“File Search”选项中，可以使用规则表达式（Regular expressions），从而能够得到更强大搜索功能。在“Containing text”输入框中还提供代码辅助功能，按“Alt+/”快捷键可以显示可供选择的规则表达式列表。

2．灵活的全局文本替换

在搜索对话框中选择“Replace”按钮，会出现如图 1-47 所示的替换对话框，该对话框新增了“Replace All in File”和“Replace All”按钮。

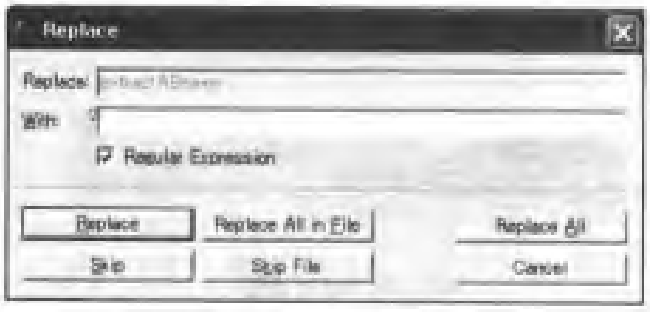


图 1-47　替换对话框

1.6.4.2　比较的新加功能

新版本中的“比较/合并”工具配置可以通过“文本比较”来调整。方法是：“Windows”

→ “Preferences” → “Workbench” → “Colors and Fonts” → “Text Compare”，可以在该选项中进行字体和颜色的更改。

1.6.4.3　CVS 的新增功能

1. 同步视图（synchronize view）

在同步视图中可以显示多个同步，如图 1-48 所示。

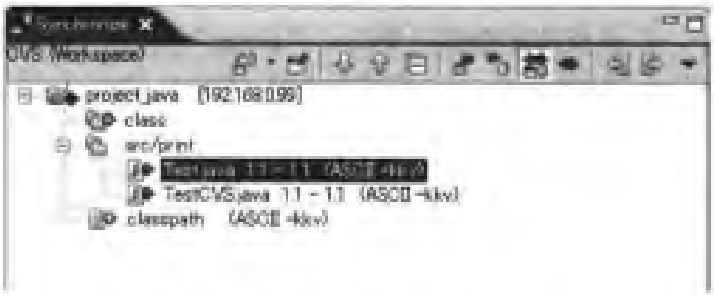


图 1-48　在同步视图中显示多个同步

2. 在新建工程或文件时，提供 CVS 检出导向

选择 “File” → “New” → “Project” → “CVS” → “Checkout Project From CVS”，单击 “Next >” 后，出现如图 1-49 所示的 “New Project” 对话框，从这个对话框中可以选择 CVS 服务器上的工程。



图 1-49　新建工程对话框

3. CVS 提交设置

在同步视图中可以通过 “commit sets” 查看提交者、提交日期和提交注释。方法是：在同步视图中选择 “Layout” → “Commit Sets”，选择后的同步视图中的情况如图 1-50 所示。

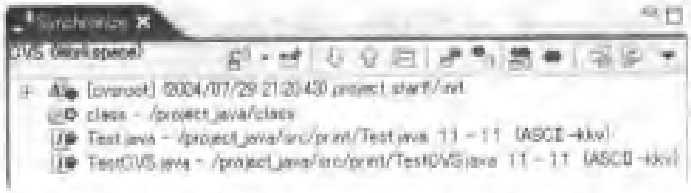


图 1-50　选择后“Commit Sets”的同步视图的情况

4. 新增 CVS 日期标记

CVS 仓库视图中，在各资源库连接中增加了“Dates”标记。在该标记上右击鼠标，然后依次选择“New”→“Date tage...”，这时会出现“Create Date Tag”对话框。在该对话框中可以调整日期标记的具体信息。

1.6.4.4　Ant 的新增功能

1. Ant 编辑器可以在构建文件中显示错误信息

新版本的 Ant 编辑器可以捕获普通 XML 的语法错误和具体的问题，并显示错误提示，如图 1-51 所示。

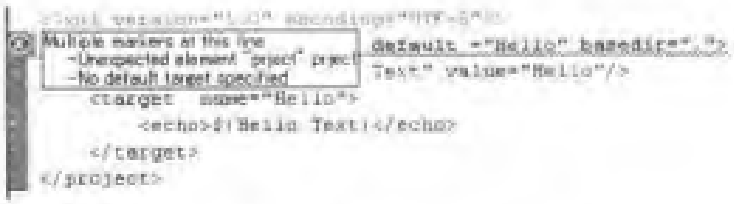


图 1-51　Ant 编辑器捕获语法错误

2. 可对 Ant 构建文件格式化处理

用户可以使用格式命令（Ctrl+Shift+F）对 Ant 构建文件进行格式化处理。

3. Ant 的格式编辑选项以及预演示功能

打开“Window”→“Preference”→“Ant”→“Formatter”，用户可以设置各种各样的编辑格式，包括 Tab 键的尺寸、最长编辑尺寸线以及预演示功能，如图 1-52 所示。

5. 构造文件模板支持功能

打开“Window”→“Preference”→“Ant”→“Template”，用户可以增加、删除和编辑 Ant 模板。

1.6.5　PDE 的新增功能

1. 一个插件，一个编辑器

新的插件编辑器是一个简单的多页编辑器，只使用这一个编辑器就可以管理插件和所有有关插件的文件（plugin.xml, build.properties 和 MAINFEST.MF 等），如图 1-53 所示。

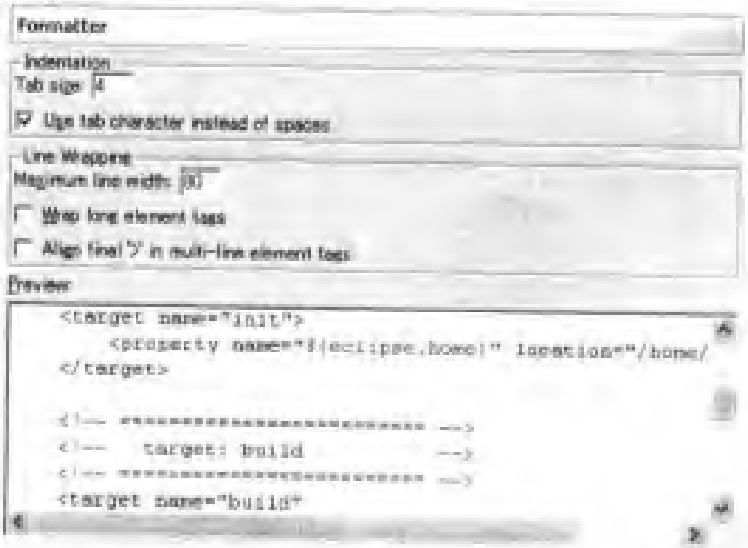


图 1-52　Ant 的格式首选项

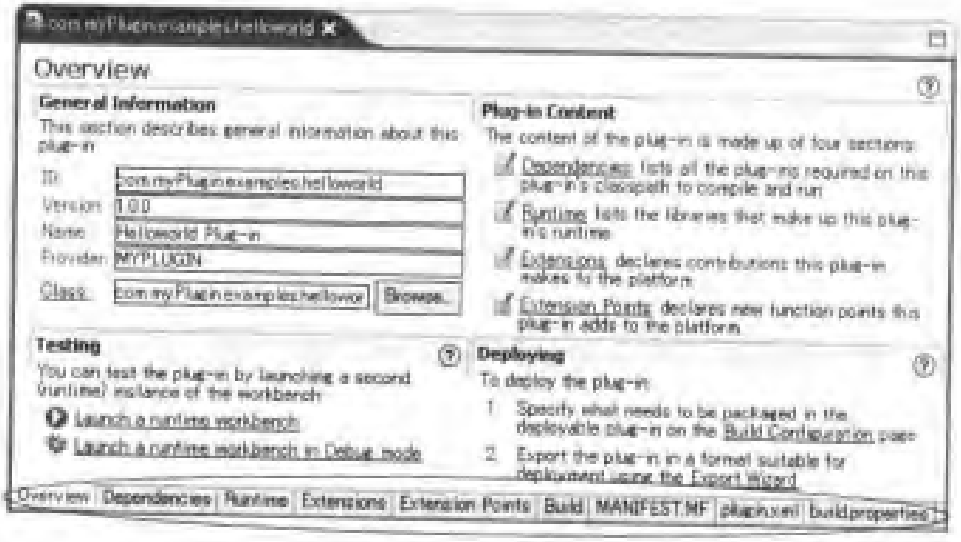


图 1-53　插件编辑器

2. 新的 PDE 构造配置编辑器

新的 PDE 提供了一个专门的构造配置编辑器，使得用户对 build.properties 文件的编辑变得简单、直观，如图 1-54 所示。

3. 改进错误日志视图

改进后的错误日志视图的工具栏可以方便地对当前的错误日志进行导入、导出、打开或删除操作。用户既可以打开外部日志，也可以重载工作区中的日志。日志中的事件可以按照“message”、“Plug-in”或者“Date”进行过滤和排序，如图 1-55 示。错误日志视图允许用户检查错误日志的详细信息，并且提供了用于在错误信息中前进或后退的按钮，以及将错误信息拷贝到粘贴板的按钮。

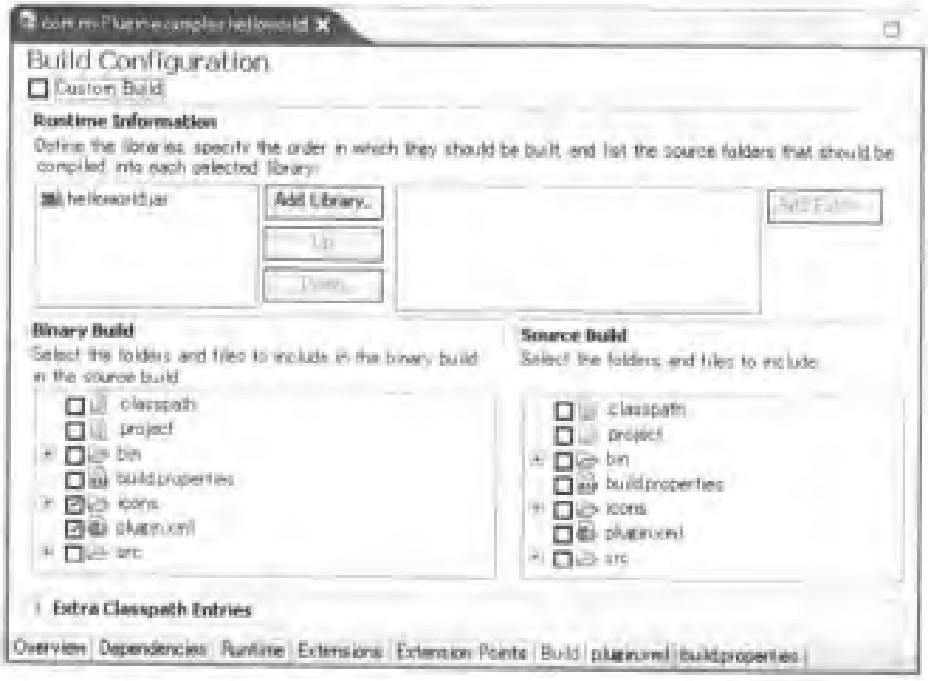


图 1-54　构造配置编辑器



图 1-55　错误日志视图

4. 新增的导出功能

插件的导出向导允许用户将插件作为目录结构导出。用户可以将插件的导出操作以 Ant 构建脚本的形式保存，这样，以后的同样操作就可以使用 Ant 脚本而不必使用导出向导了，如图 1-56 所示。

5. 扩展了 Java 搜索范围和源代码查找

“Add to Java Search”用来扩展 Java 搜索的范围，使 Java 搜索范围除了工作区中的项目之外，还扩展到组成用户目标平台的外部插件中的库文件。实现这一功能是通过在插件上单击右键并选择“Add to Java Search”来完成的。当用户升级插件的时候 PDE 会自动更新库文件。这样做的优点是确保调试器可以自动定位相关的源代码，如图 1-57 所示。



图 1-56　插件导出向导



图 1-57　“Add to Java Search”选项

本章小结

Eclipse 是一个非常优秀的集成开发环境，本章仅对 Eclipse 的一些基本概念进行了简单的介绍。Eclipse 3.0 版本是目前最新的版本，但是，本书将以 Eclipse 2.1 作为主要的工具，因为这个版本是目前最成熟和稳定的。“插件”是 Eclipse 最吸引人的地方，也是 Eclipse 的灵魂之处，当你在编程的过程中遇到不顺心的时候，去找一些有助于你工作的插件，又或者亲手制作一个“得心应手”的插件，或许能够“事半功倍”呢？

在下一章中，将对 Eclipse 平台的工作原理、体系结构、具体使用方法等做更加深入的介绍。

第 2 章　Eclipse 平台

Eclipse 平台为工具提供者提供一种机制，使得各种工具能够无缝集成。这些机制通过 API 接口、类和方法来体现。Eclipse 平台负责处理基本环境的后台工作，并提供标准的用户导航模型。同时，Eclipse 平台还提供了开发新插件的框架（如 PDE），使开发插件工作变得简单易行。本章将详细介绍 Eclipse 平台及其主要部件，以及 Eclipse 的使用方法等，具体内容包括：

- Eclipse 平台技术概述，介绍平台的体系结构及各组成部分。
- Eclipse 工作台的基本操作，介绍工作台的组成部分及工作台的基本使用方法。
- Eclipse 资源管理，分别介绍在工作区和工作台中的资源管理方法。
- 定制 Eclipse 工作台，介绍工作台的各种定制方法。
- 工作台的使用技巧。

2.1　Eclipse 平台技术概述

Eclipse 平台是在其上创建插件扩展的核心框架和服务的名称，它提供了在其中装入和运行插件的运行时。Eclipse 平台本身并不是一个完整的产品，平台的直接客户是工具构建器或者是 ISV，它们以插件的形式集成到 Eclipse 平台中，从而使它们能够物尽其用。

Eclipse 平台的设计和构建是为了满足下述需要：

- 支持多种应用开发工具的构建，如 Java 程序开发工具（JDT）、C 程序开发工具（CDT）。
- 支持无限制的工具提供者，包括 ISV。
- 支持任意的文件格式（例如，HTML，Java，C++，JSP，EJB，XML，GIF）。
- 帮助不同工具提供者的无缝集成。
- 支持 GUI 和基于非 GUI 的应用开发环境。
- 支持在不同操作系统上运行，包括 Windows 和 Linux。

2.1.1　平台的体系结构

Eclipse 平台定义了一个开放式体系结构，以便使每个插件开发小组都能够专注于它们擅长的领域。让资源库专家构建后端，让可用性专家构建最终用户工具。如果平台设计好了，就可以添加重要的新功能部件和集成级别，而不会影响其他工具。

Eclipse 平台是构建在发现插件模型、集成插件模型和运行插件模型的机制上的。工

具提供者编写的工具作为一个独立的插件，它能运行工作区里的文件，并且在工作台上显示出它特定的用户界面外观。当平台被启动时，Eclipse 显示给用户的是一个由可利用插件集合组成的集成开发环境。

Eclipse 平台分为核心和用户界面。用户界面中的任何事项都需要一个窗口系统，而核心的事项则是隐藏在后台运行的。Eclipse 平台的用户界面部分称为工作台，核心部分称为平台核心或简称核心。因此，Eclipse 平台就是工具构建器围绕着其构建工具插件的核心。Eclipse 平台的结构如图 2-1 所示。

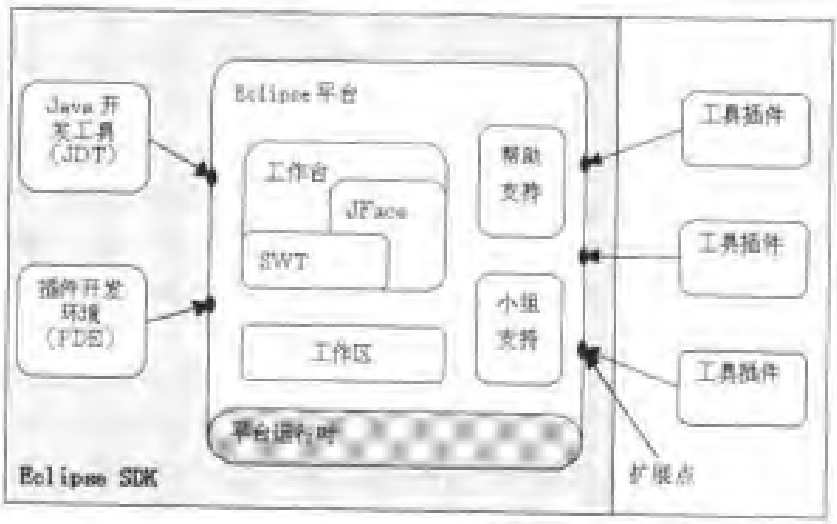


图 2-1　Eclipse 平台的体系结构

从最终用户的观点来看，Eclipse 平台使用工作台模型来集成各种工具，使用扩展点来将用户开发的工具插件插入到平台中。Eclipse 平台正是根据扩展点这一概念来建立的，扩展点是系统中严格定义的一些位置，同时其他工具（也称为插件）可以在此添加功能。

平台中的每个主要子系统本身是由一组插件来构成的，这些插件实现一些重要功能并定义扩展点。Eclipse 系统本身是通过向第三方插件供应商提供扩展点来构建的，插件供应商可以通过这些扩展点来添加各种功能。插件可以定义它们自己的扩展点，或者只是将扩展的功能添加至其他插件的扩展点上。

平台子系统通常将可视功能部件添加到平台中，并提供 API 来扩展它们的功能。这些组件中的某些组件提供与扩展点不直接相关，但可用来实现扩展的附加类库。例如工作台用户界面提供 JFace 用户界面框架和 SWT 小窗口工具箱。

Eclipse SDK 中包含基本平台以及两个主要工具，它们对于插件开发很有用。Java 开发工具（JDT）实现了全功能的 Java 开发环境。插件开发者环境（PDE）添加了专门工具，使开发新插件或者扩展原有插件功能更加方便。

底层的平台运行时使用同一扩展模型来允许插件开发者添加其他文件类型和定制安装的支持，例如，Web 服务器、工作组服务器和资源库等。每个工具插件是由公共平台资源模型来协调的。

平台为用户提供了使用工具的常见方法，并对它们利用插件创建的资源提供集成管

理。插件开发者也因此体系结构而受益。平台负责管理不同运行时环境的复杂事务，例如，不同操作系统或工作组服务器环境。插件开发者可以专注于完成他们的特定任务，而不用担心这些集成问题。

下面就对平台的插件体系结构以及平台各主要组成作详细介绍。

2.1.2 平台运行时

平台运行时（Platform Runtime）是 Eclipse 平台核心运行时的简称，是平台核心中最低级别的部分，负责管理插件注册表和插件。平台核心运行时不包括工作区和资源（它们在“资源”插件中）。平台运行时实现运行时引擎，它启动平台基体，并动态地发现插件。插件是一种结构化组件，它使用插件清单文件（plugin.xml）来向系统描述它自己。

平台负责维护已安装的注册表以及它们提供的功能。功能是使用常见的扩展模型来添加的。扩展点是系统中严格定义的功能点，也可以由插件来扩展。当插件向扩展点添加实现时，就认为它向平台添加了扩展。插件可以定义它们自己的扩展点，以便让其他插件可以与它们紧密地集成在一起。

扩展机制是将功能添加到平台和其他插件的惟一方法。所有插件都使用相同的机制。随 Eclipse SDK 提供的插件（如 JDT 和 PDE）在它们的实现中不使用任何专用机制。

扩展通常是使用由 Java 编写的平台 API。而某些扩展点也接受作为平台可执行文件、ActiveX 组件来提供的扩展，或者是用脚本语言开发的扩展。通常，只有完整平台功能的子集可用于非 Java 扩展。

最终用户不应该为已安装但未使用的插件消耗内存或性能，因此运行时将插件安装和添加到注册表中但不激活插件，除非是通过用户活动请求了插件所提供的功能时才会去激活该插件，这样一来就大大降低了内存消耗，提高了系统性能。

Eclipse 平台的更新管理器可以下载和安装新的功能部件，或者升级现有功能部件的版本（功能部件是一组相关的插件）。更新管理器在 Eclipse 平台下次被启动时就会构造一个可利用插件的新配置。如果升级或者安装的结果证明是不令人满意的，用户能回滚到原来的配置中。

2.1.3 工作区

工作区（worlspace）是用于管理 Eclipse 平台中的资源的，它定义 API，用以创建和管理由工具插件生成的并保存在文件系统中的资源（包括项目、文件和文件夹）。注意：工作区与资源都是平台的可选部分，某些平台的配置可能不具有工作区。

各种插入到 Eclipse 平台上的插件能够运行用户工作区内的常规文件。工作区包含一个或多个顶层项目，在那里每个项目映射到文件系统中用户指定目录下。在工作区内，不同的项目可以映射到不同的文件系统目录。在缺省情况下，所有的项目映射到一个叫“workspace”目录下的相邻子目录下。

1．源和工作区

用户的数据文件的中央集线器称为工作区。可以认为平台工作台是一种工具，它允许用户浏览和处理工作区。资源插件提供了用于创建、浏览和处理工作区中的资源 API。工作台使用这些 API 来为用户提供此功能，其他插件也可以使用这些资源 API。

资源 API 允许用户用特殊信息来标记资源，了解对资源的更改，实现对资源执行转换的专用构建器，并将特定于插件的信息与资源一起保存。每个项目包含由用户创建和操作的文件，底层操作系统的标准程序和插件可直接访问在工作区内的所有文件。集成到平台上的插件提供 API 来处理工作区资源。工作区资源由适当的对象来描述，以便第三方能扩展他们的行为。

从基于资源的插件的观点来看，只有一个工作区，并且只要插件在运行，它就始终是打开的，以使处理事务。当激活资源插件时，工作区会自动打开，当关闭平台时，平台也会关闭。如果你的插件需要资源插件，则资源插件将在你的插件之前启动，而工作区将是可用的。

工作区包含资源集合。在用户的透视图中，有三种不同类型的资源：项目、文件夹和文件。项目是任意数目的文件和文件夹的集合。它是用来组织与特定区域相关的其他资源的容器。文件和文件夹就像文件系统中的文件和目录。文件夹包含其他文件夹或文件。文件包含任意顺序的字节。其内容不是由平台解释的。

工作区的资源组织成树形结构，项目位于顶部，而文件夹和文件在下面。特殊资源、工作区根目录资源充当资源树的根目录。工作区根目录是在创建工作区时在内部创建的，只要工作区存在，工作区根目录就存在。工作区可以有任何数目的项目。

2．资源插件

平台中提供的其中一个基本插件是资源插件（称为 org.eclipse.core.resources）。资源插件提供服务，以帮助访问用户正在使用的文件。资源管理插件定义常见的资源模型，该模型用来管理工具插件的助诊文件。插件可以创建和修改项目、文件夹和文件，以及定义专用类型的资源。

3．项目性质机制

项目性质机制允许插件将项目标记为特定类型的项目。例如，使用“网站性质”来将网站的静态内容添加到项目中，使用“Java 性质”来将特定于 Java 的行为添加到项目中。项目性质机制是开放的，插件可以声明新的项目性质，并且为配置具有此项目性质的项目提供代码。项目性质由插件定义，并通常当用户执行一些由插件定义的操作时按项目进行添加或除去。一个项目可以具备多个性质，这样就为插件提供了一种方式，即可以分享一个项目而不需要彼此了解。

4．跟踪工作区的资源更改机制

为使偶然丢失文件的危险减到最小，平台提供一个历史记录机制来允许插件跟踪工作区资源的变化，将对资源所作的更改通知给感兴趣的用户。

要跟踪资源的更改，必须向工作区注册资源更改侦听器。在执行了对工作区资源的修改之后，将通知侦听器。修改资源的资源 API 方法将触发这些事件，作为描述行为的

部分。资源 API 方法的方法注释显式地声明它是否触发了资源更改事件。通常，创建、删除或更改资源的方法将触发这些事件，读取资源不会触发资源更改事件。

如果需要同时修改一些资源，则可以对资源更改 API 调用进行批处理，以便对整组更改只发送一个资源更改事件。当完成指定的可运行程序（runnable）时，只把资源更改事件发送给已注册的侦听器。在可运行程序中创建的所有附加（嵌套）可运行程序将被看做是父代批处理操作的一部分，而在那些可运行程序中所做的资源更改将出现在父代的资源更改通知中。应当尽可能地对更改进行批处理，以限制广播许多微小更改所造成的开销。如果未能使用批处理，则可能会使系统中的资源更改通知和自动构建暴增。

每当对工作区进行更改（或一批更改）时，就会发送资源更改事件。另外，对于某些特定工作区操作也会发送资源更改事件。资源更改事件描述在工作区中已经发生的更改（或更改集合）的细节。事件中包含资源变化，它描述更改的净效应。例如，如果用户在一次批处理中添加了资源，稍后又删除了该资源，则该资源将不会出现在变化中。资源变化树描述下列类型的更改：

- 已经创建、删除或更改的资源。如果已经删除（或添加）了文件夹，则资源变化将包含该文件夹以及该文件夹中包含的所有文件。
- 已经使用 API 移动或重命名的资源。
- 已经添加、除去或更改的标记。将标记修改看作是工作区修改操作。
- 已经修改的文件。在资源变化中标识了已更改的文件，但是用户无法访问资源变化中的文件的先前内容。

5. 作区的标记机制

工作区为注释资源提供一个标记机制，以便向用户通知有关问题或其他信息。标记用于记录各种注释，例如编译程序错误信息、任务、书签、搜寻内容和调试断点。标记机制是开放的，平台资源 API 定义用于创建标记、设置标记值和利用新的标记类型来扩展平台的方法。在平台管理标记时，标记的创建、除去和属性值是由插件来控制的。

标记应是小的轻量级对象。在单个项目中可以有成百上千个标记。例如，Java 编译器使用标记来对它在源代码中发现的每个问题作出标志。平台将抛弃与已删除的资源相连的标记，而在标记不再适用于仍然存在的资源时，插件将负责除去失效的标记。

2.1.4 工作台和用户界面工具箱

工作台（workbench）是一个高级用户界面（UI）框架，Eclipse 平台用户界面构建在工作台之上，它为用户提供了一个整体架构，并且提供了一个可扩展的用户界面。工作台的 API 和执行是由以下两个工具箱来构建的。

- 标准小窗口工具箱（简称 SWT）：一个与本机底层窗口系统紧密集成的小窗口工具箱和图形库，但是却提供独立于操作系统的可移植的 API。
- JFace：一个使用 SWT 实现的一个用户界面工具箱，它提供用于处理常见的用户界面编程任务的类，从而使工具开发者专注于实现特定插件的功能。

下面具体介绍这几个部分。

2.1.4.1　SWT

SWT 是 Java 开发者的小窗口工具箱，它提供可移植的 API，并与本机底层窗口系统平台 GUI 紧密集成。许多低级别的用户界面编程任务是在 Eclipse 平台的较高层处理的。例如，用户界面添加项的 plugin.xml 标记指定菜单和工具栏内容，而不需要进行任何 SWT 编程。另外，JFace 查看器和操作实现了应用程序与小窗口之间的普通交互作用。

Java Swing 工具箱试图通过提供高级别小窗口（例如，树和丰富文本）的非本机实现来解决此问题。这提供了许多功能，但使得在 Swing 中开发的应用程序因为不同而显得很突出。平台外观模拟层帮助应用程序看起来更像平台，但是用户交互有很大不同。这使得很难使用模拟的工具箱构建应用程序与为特定操作系统平台专门开发的包装好的应用程序竞争。

SWT 通过定义在所有受支持的平台上提供的常见可移植 API 来解决此问题，并尽可能使用本机小窗口在每个平台上实现该 API。这允许工具箱在所有平台上维护一致的编程模型时，能立即反映底层操作系统 GUI 外观中的任何更改。

SWT 具有可移植性和平台集成的功能，Java AWT（抽象窗口工具箱）为较低级别的小窗口（例如，列表、文本和按钮）提供了平台集成的小窗口，但是没有提供对较高级别的平台组件（例如，树或丰富文本）的访问权。这强制应用程序开发者进入“最小公分母”情况，在这种情况下，他们只能使用所有平台上都可用的小窗口。“最小公分母”问题是由 SWT 按以下几种方式来解决的：

- 在未提供任何本机支持的平台上，可以对在所有平台上都未提供但是通常对工作台和工具插件很有用的功能部件进行仿真。例如，OSF/Motif 2.1 小窗口工具箱不包含树形小窗口，因此 SWT 在 Motif 2.1 上提供了仿真树形小窗口，它是与 Windows 本机实现相兼容的 API。
- 可以从 SWT 中省略在所有平台上都未提供但是未广泛使用的功能部件。例如，Windows 提供了实现日历的小窗口，但是在 SWT 却没有提供。
- 仅在相关的平台上提供了特定于平台的功能部件，例如，ActiveX 集成。特定于平台的功能部件分成单独的包，它在包中明确表示平台名。

平台集成严格说来并不是外观问题。通过紧密集成，能够与本机桌面功能部件（例如，拖放）交互作用、与操作系统桌面应用程序集成，以及使用利用操作系统组件模型（例如，Win32 ActiveX）开发的组件。

通过提供本机操作系统开发者熟悉的实现，同时在代码本身中获得了一致性。SWT 为每个平台提供了采用 Java 的不同实现，而不是隐藏本机 C 语言代码中的操作系统区别或试图在 Java 实现中构建可移植和非可移植的层。

一个重要的实现规则是使用 C 语言的本机程序与对操作系统的调用进行一一映射。Windows 程序员将立即识别 Windows 上的 SWT 工具箱的实现，因为它使用直接映射至在 C 语言中使用的系统调用的本机程序。没有任何“平台幻数”隐藏在 C 语言代码中。平台开发者可以查看代码，并确切地知道哪些平台调用是由工具箱执行的，这极大地简

化了调试。如果在调用本机方法时发生了故障，则从 C 语言代码中使用相同参数来调用平台 API 将产生相同的故障。

2.1.4.2 JFace

JFace 是一个用户界面工具箱，它提供很难实现的、用于开发用户界面功能部件的 helper 类。JFace 在原始的小窗口系统的级别之上运行。它提供用于处理如下常见的用户界面编程任务的类。

- 查看器：负责处理填充、排序、过滤和更新小窗口等最辛苦的工作。
- 操作和添加项：介绍用于定义用户操作的语义，并指定在何处提供它们。
- 图像和字体注册表：提供用于处理用户界面资源的常见模式。
- 对话框和向导：定义用于与用户进行复杂交互的框架。

JFace 使你可以专注于实现特定插件的功能，而不必花费精力来处理底层小窗口系统或者解决几乎在任何用户界面应用程序中都很常见的问题。

1. JFace 和工作台

何处是 JFace 结束而工作台开始的位置呢？有时候界线并不是很明显。通常，JFace API（来自于包 org.eclipse.jface.*）与工作台扩展点和 API 无关。可以想像，根本不必使用任何工作台代码就可以编写 JFace 程序。

工作台使用 JFace，但是又试图尽可能减少依赖项。例如，工作台部件模型被设计为独立于 JFace。我们很早就知道可以直接使用 SWT 小窗口来实现视图和编辑器，而不必使用任何 JFace 类。工作台尽可能保持“JFace 中立”，允许程序员使用他们觉得有用的 JFace 的部件。实际上，在工作台的大多数实现中都使用了 JFace，并且在 API 定义中引用了 JFace 类型。

2. JFace 和 SWT

SWT 和 JFace 之间的界线是很明显的。SWT 根本不依赖于任何 JFace 或平台代码。许多 SWT 示例说明可以如何构建独立的应用程序。

JFace 用来在 SWT 库顶部提供常见的应用程序用户界面功能。JFace 并不试图“隐藏”SWT 或者替换它的功能。它提供一些类和接口，以处理与使用 SWT 来对动态用户界面编程相关联的许多常见任务。读者通过考察视图及它们与 SWT 小窗口之间的关系，就可以更清楚地了解 JFace 和 SWT 之间的关系。

3. 查看器类

利用 SWT 小窗口来直接实现工作台用户界面添加项（例如，视图、编辑器、向导和对话框），而查看器机制允许用户在创建小窗口时仍然可以使用模型对象。如果直接使用 SWT 小窗口，则需要将对象转换成 SWT 期望的字符串和图像。查看器充当 SWT 小窗口上的适配器，处理用于处理小窗口事件的常见代码，否则，将需要由用户自己来实现。

JFace 为 SWT 中的大多数重要的小窗口提供查看器。查看器通常用于列表、树、表和文本小窗口。每个查看器都有相关联的 SWT 小窗口。可以用两种方式创建一个小窗

口，通过在查看器构造函数中提供父组合体来隐式地创建此小窗口，或者先创建一个小窗口，然后将它提供给其构造函数中的查看器，从而显式地创建它。

注意：可以使用查看器来同时为工作台视图和编辑器提供实现。术语“查看器”并不意味着它们只能用于实现视图。例如，文本查看器用于实现许多工作台和插件编辑器。

4．操作类

操作类允许用户定义用户命令而不考虑它们在用户界面中的表示。这使用户能够灵活地更改操作在插件中的表示，不必在选择某个命令时更改实际执行该命令的代码。操作表示可以由最终用户的命令来触发。操作通常与按钮、菜单项和工具栏中的项相关联。尽管操作本身不放置在用户界面中，但是它们确实具有面向用户界面的属性，例如，工具提示文本、标号文本和图像。这允许其他类为操作的表示构造小窗口。

当用户触发用户界面中的操作时，就会调用该操作的运行方法来执行实际工作。运行方法中的常见模式是查询工作台选择并处理所选的对象，另一种常见模式是在选择操作时启动向导或对话框。

5．添加项类

添加项表示操作的用户界面部分。更准确地说，它表示添加到共享用户界面资源的项，例如，菜单或工具栏。添加项类用来管理表示命令的实际用户界面项。用户可以在某些工作台和 JFace API 中看到添加项类，而不需要对其进行编程。

添加项管理器表示将在用户界面中提供的添加项的集合。可以通过使用已命名的添加标识来添加和插入添加项，以便按照相应的次序来放置这些添加项。还可以根据标识来查找添加项和除去个别项。添加项知道如何利用表示该添加项的相应 SWT 项来填充该 SWT 小窗口。JFace 为菜单、工具栏和状态行提供了添加项管理器。

2.1.4.3　工作台

从上面的介绍中可知，SWT 和 JFace 都定义为用户界面。SWT 是与本机窗口系统对话的、与操作系统平台无关的低级别方法。JFace 是中级用户界面框架，负责构建复杂的用户界面部分（如属性查看器）。与 SWT 和 JFace 这两个通用用户界面工具箱不同，工作台是一个高级用户界面框架，它提供具有个性的 Eclipse 平台的用户界面，用于构建由插件构建的复杂用户界面的产品。工作台是在 Jface，SWT 和平台核心的基础上构建的。Java tooling 是一个在工作台的基础上构建的 Java IDE。

Eclipse 平台的用户界面是由编辑器，视图和透视图组成的。站在用户的角度来看，一个工作台窗口包含了视图和编辑器。透视图是根据自己不同的选择来布局视图和编辑器的。

1．编辑器

编辑器允许用户打开、编辑和保存对象。在编辑器中执行的修改遵循“打开-保存-关闭”这一生命周期模型，这样一来就更像基于工具的文件系统的生命周期，从而更加紧密地与工作台整合。当编辑器被激活时，工作台的菜单和工具条中就会包含适用于该编辑器的操作。平台提供一个标准的文本编辑器资源，更多具体应用的编辑器可以由插

件来提供。

2．视图

视图提供关于用户正在工作台上工作的一些有关对象的信息。视图可以通过提供有关被编辑资料的信息来帮助编辑者。例如，一个标准的大纲视图显示了活动的编辑器内容概述。视图可能通过提供关于目前选择的物体的信息增大其他视图。例如，标准的属性视图提供在另一视图选择的对象的属性。与编辑器相比较，视图有一个更简单的生存周期：在视图内修改（例如修改一个属性值）保存后，在相关的用户界面中其变化立即被反映出来。平台提供了几种标准的视图，另外的视图也是由插件来提供的。

3．透视图

一个工作台窗口可包含多个独立的透视图，但在同一时刻仅仅其中一个透视图是可见的。每个透视图有它自己的视图和编辑器。每个透视图都提供了一组功能，目的在于完成特定类型的任务或使用特定类型的资源。例如，“Java”透视图将编辑 Java 源文件时常用的视图组合在一起，而“调试”透视图包含将在调试 Java 程序时使用的视图。透视图定义工作台窗口中各视图的初始设置和布局。用户在工作台中工作时，可以快速频繁地在各个透视图之间切换。用户还可以很轻松地重新布局或定制一个透视图，使它更加适合于特定的任务。平台提供了一些标准透视图，例如，资源导航、在线帮助、小组支持等透视图，其他透视图是由插件来提供的。

4．工具插件

工具是以一个严格定义的方式集成到用户界面，扩展点允许工具来扩展工作台，包括：

- 增加新的编辑器类型；
- 增加新的视图类型；
- 增加新的透视图，这里包括扩展原有的视图来适合新用户任务的需要。

平台标准的视图和编辑器全都适用于这些机制。

工具也可以扩展现有的编辑器、视图和透视图：

- 在一个现有视图的菜单和工具条中添加新的操作；
- 当一个现有编辑器被激活时，给工作台菜单和工具条添加新的操作；
- 在现有视图或编辑器的弹出内容菜单中添加新的操作；
- 在一个现有的透视图中增加新的视图、操作设定和快捷方式。

平台负责整个工作台窗口和透视图管理。需要时，编辑器和视图会被自动初始化，当不再需要时，编辑器和视图就会被关闭。操作所显示的标签和图标被放在插件列表里，以便工作台在没有激活插件的情况下也能创建菜单和工具条。工作台直到用户想使用插件提供的功能时才会去激活插件。

一旦某个视图或编辑器成为透视图中的活跃部分，它就能够使用工作台提供的跟踪活动和选择的服务。该服务能够跟踪在透视图内活动的视图和编辑器，报告活动和停用事件给注册监听者。视图或编辑器也可以通过选择服务来注册，作为一个选择项。选择服务将更新的事件提供给所有已经注册的选择，因此当前活动的编辑器或视图领域内的

对象就会被通知给标准的属性视图。

2.1.4.4　用户界面集成

用 Java 编写的插件工具通过平台 APIs 可以与平台进行最高级别的集成。从另一个极端讲，为了能与用户通信，从平台内部启动外部工具时必须要打开他们自己的独立窗口，并且能够通过基础文件系统访问用户数据，这样的集成就会因此而非常松散，特别对用户界面的水平上的集成更是如此。在一些环境里，Eclipse 平台也支特在这两种极端之间的各种级别的集成。

在内部编辑器的情况下，可实现工作台窗口与编辑器部件之间的紧密集成。工作台菜单和工具栏预装入了许多公共操作（如剪切、复制和粘贴）。活动的部件、视图或编辑器应会提供这些操作的实现。内部编辑器还可定义出现在工作台窗口中的新操作，不过只有当编辑器活动时，这些操作才会出现。

工作台与外部编辑器之间的集成则松散一些。在这种情况下，工作台可以启动编辑器。但自此以后，除了通过文件系统之外，它再没有任何办法确定外部编辑器的状态，也不能与外部编辑器件合作。

2.1.5　小组支持

Eclipse 平台基于的概念是让用户在文件系统中创建和编辑资源。小组插件定义附加的 API，允许插件集成版本和配置管理资源库的功能。资源库提供的功能将从根本上影响用户工作流程，这些功能包括：附加的步骤检索文件、与本地内容比较其内容、为它们进行版本控制，并将更新的文件返回至资源库。

小组插件 API 被构造为被动的，目的是允许资源库插件提供的功能不变，不影响用户的工作流程，这样熟悉其产品的用户就可以用相似的方式使用平台，这是通过允许小组插件干预控制资源操作的基本 hook 来实现的。例如，当用户保存文件时，资源库提供程序可以将其保存在资源库中，询问用户是否应将它登入资源库，或将它与资源库中的内容进行比较。

用户界面支持也被构造成被动的。小组提供程序操作、首选项和属性的占位符由小组用户界面插件定义，但要由小组插件提供程序来定义这些用户界面元素。小组用户界面插件还包括一个简单的可扩展配置向导，它使用户可将项目与资源库连接。插件可为此向导提供内容，该内容让用户指定资源库特定信息。

多个资源库提供程序可以在平台中“和平共处”。实际上甚至可以安装同一资源库的不同客户机实现。例如，一个可以安装为专家设计的 CVS 客户机，另一个为新用户安装。

在本书的后续章节中，将会给读者详细介绍有关小组支特的内容，包括介绍什么是 CVS、如何利用 CVS 来进行版本管理等。

2.1.6 帮助支持

联机帮助系统让用户能够浏览、搜索和打印系统文档。文档组织成与书籍类似的信息集。帮助系统还提供了用于按关键字查找所需的信息的文本搜索引擎和用于查找描述你正在使用的特定功能的信息的上下文相关帮助。

2.1.6.1 帮助系统的使用

1. 帮助浏览器

在工作台的菜单中选择“帮助”→“帮助内容”首次打开帮助浏览器时，显示的第一个视图称为“内容”，如图 2-2 所示。“内容”视图显示产品文档的各种分组。单击其中一个链接可以展开该组文档的导航树。浏览该树并单击主题以显示其内容。



图 2-2 帮助浏览器

2. 搜索

如果想要在联机帮助中查找特定的信息片段，可以使用“搜索”工具。帮助系统将搜索整个信息集或仅搜索信息集的某个部分来查找符合指定条件的主题。

3. 上下文相关帮助

如果正在完成某个任务但遇到了你不了解的界面部分，则使用弹出信息。通过单击所讨论的界面小窗口或者使用 Tab 键来使该小窗口处于焦点中，然后按 F1 键。弹出信息将提供有关界面的信息以及指向更多信息的链接。如图 2-3 所示的画面，当激活导航器视图后，按 F1 键就会弹出一个黄色的对话框，来解释这个部分的作用，例如，图中弹出框中解释了导航器视图的作用是“显示工作区中的所有资源”，并提供了相关信息的链接，例如，如果用户想了解更多的内容，可以单击图中“视图”和“导航器视图”的链接。



图 2-3 上下文相关帮助

4. 帮助的其他使用技巧

- 帮助书签：可以保留你自己用来标明帮助书籍中页数的书签列表。使用“帮助”浏览器的工具栏上的书签文档按钮来创建书签。书签将显示在书签选项卡中。
- 帮助工作集：帮助工作集允许将搜索范围缩小到一本书中的一节。工作集将从一个会话保持至下一个会话，并且可在工作台帮助搜索中使用或者从“帮助”浏览器中使用。
- 配置帮助浏览器：如果用户有自己喜欢的 Web 浏览器，也可以通过“窗口”→“首选项”→“帮助”→“帮助服务器”来配置自己想要的 Web 浏览器，而不是系统默认的浏览器。

2.1.6.2 帮助系统的插件

帮助系统的插件提供了平台优化的帮助、Web 在线帮助以及用户文档集成的功能。它定义了一些扩展点，插件可以使用这些扩展点添加帮助或者添加其他插件文档作为可浏览的书籍。文档 Web 服务器包含特殊的功能，它允许帮助插件通过使用逻辑的、基于 URL 的插件的方式来引用文件，而不是以往的通过文件系统 URL 来引用文件。帮助系统的插件还提供了其他功能部件来集成产品级别文档配置中的帮助主题。

Eclipse 平台的帮助功能为用户提供了原始构建块来构造和添加平台文档。它不会限制文档的结构或粒度。用户可以为文档选择符合需要的工具和结构。帮助系统的插件允许用户使用目录文件描述平台的文档结构。平台使用 org.eclipse.help.toc 扩展点添加插件的联机帮助，可以作为代码插件的一部分来添加联机帮助，也可以在它自己的文档插件

中单独提供联机帮助。

帮助系统插件的高级功能包括具有参考链接（称为弹出信息）的上下文相关帮助和从文档调用平台代码的能力。帮助浏览器允许你查看、打印和搜索文档。演示这些帮助添加项的最佳方法就是创建文档插件。

2.2 Eclipse 工作台

2.2.1 工作台概述

工作台（WorkBench）是一个桌面开发环境，对于使用 Eclipse 的用户来说，它的外部表现为“工作台”窗口。

“工作台”的目标是通过创建、管理、引导工作台资源提供一个公共的规范来获得无缝工具和受控的公开。如图 2-4 所示，“工作台”窗口主要是由以下几个部分组成：

- 标题栏
- 菜单栏
- 工具栏（tool bar）
- 透视图（perspective）

其中，工具栏分为三种工具栏：

- 主工具栏（main toolbar）
- 视图工具栏（view toolbar）
- 快捷方式栏（shortcut bar）

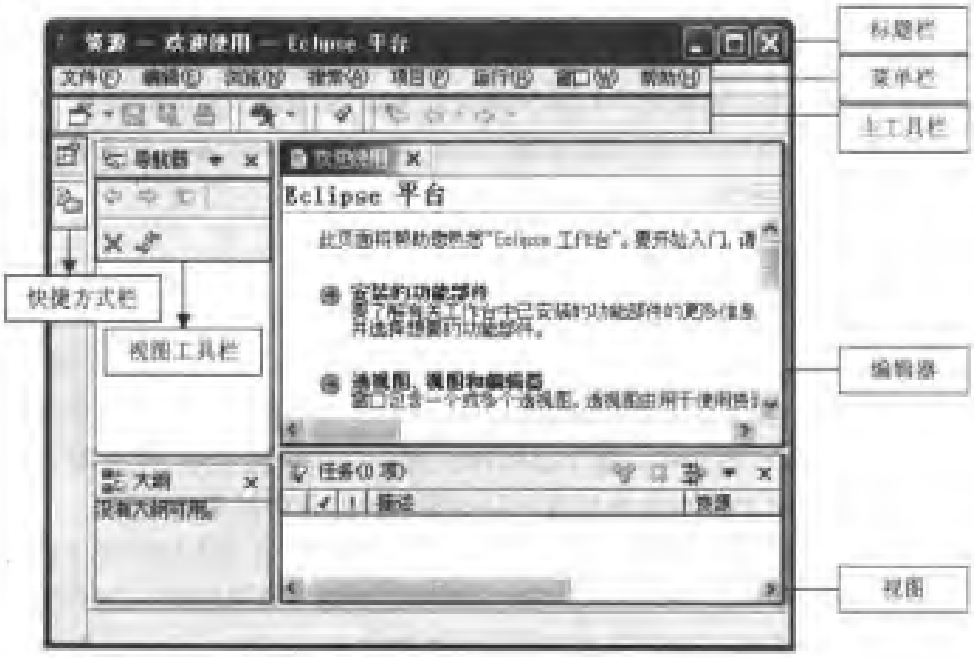


图 2-4 工作台窗口

透视图又是由以下两个部分组成：

- 视图（view）
- 编辑器（editor）

下面对这些部分的概念和操作进行详尽的介绍，通过这些介绍读者会对 Eclipse 的工作台有一个具体的概念，同时，可以灵活、熟练地在“工作台”窗口中进行项目开发。

2.2.2　透视图

在每个“工作台”窗口中都可以含有一个或多个透视图（perspective）。每个透视图有多个视图和编辑器组成。一个透视图定义了每个视图在“工作台”窗口中的初始设置和布局。而且，每个透视图共享同一套编辑器。

比如，想要编辑一个 Java 源程序，那么就需要打开一个 Java 透视图，Java 透视图包含有编辑 Java 源程序所需要的常用视图和 Java 源程序的编辑器。当需要测试该程序时，则要切换到 Debug 透视图，Debug 透视图含有测试所需要的常用的视图和 Java 源程序的编辑器。

1. 打开透视图

打开一个透视图有两种方法：

- 选择菜单栏中的“窗口（W）”→“打开透视图”→“其它（O）...”，在打开的窗口中就可以选择要打开的透视图了。
- 可以通过“快捷方式栏”打开一个透视图。如图 2-5 所示，“快捷方式栏”最上面的按钮是“打开透视图”下拉按钮，在该按钮下面的几个按钮是已打开的透视图的切换按钮。要想打开一个透视图只要单击“打开透视图”下拉按钮，就会出现“透视图下拉菜单”如图 2-6 所示。在下拉菜单中选择“其它（O）...”，在打开的窗口中选择要打开的透视图。

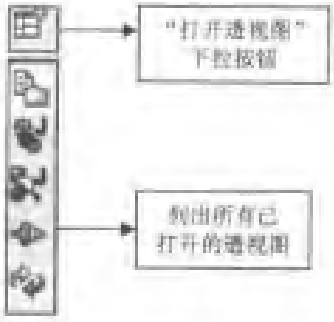


图 2-5　透视图快捷方式栏

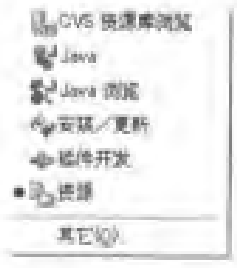


图 2-6　透视图下拉菜单

2. 已打开透视图之间的切换

在已打开透视图之间进行切换可以用下面三种方法：

- 如图 2-5 所示，在“快捷方式栏”列出了已打开的透视图，只要用鼠标单击已打开的透视图，就可以完成透视图之间的切换。

- 如图 2-6 所示，在“其它（O）”的上面已经列出了已打开的透视图，只要用鼠标单击已打开的透视图，也可以完成透视图之间的切换。
- 选择菜单栏中的“窗口（W）”→“打开透视图”。在下拉菜单中列出了已打开的透视图，再单击已打开的透视图，同样可以完成透视图之间的切换。

3. 关闭透视图

关闭透视图很简单，在“快捷方式栏”中右击要关闭的透视图的图标，这时，下拉菜单中会出现两个选项：关闭、全部关闭。其中，“关闭”是指关闭所选的透视图；“全部关闭”则是将已打开的透视图全部关闭。

2.2.3 视图

视图（view）支持编辑器而且提供可供选择的信息和方法来得到工作台的信息。

1. 打开视图

选择菜单栏中的“窗口（W）”→“显示视图（V）”。在下拉菜单中可以选择想要打开的视图。比如，打开“导航器”视图，单击“导航器”，则“导航器”视图就被打开了。如图 2-7 所示，在“导航器”视图标题栏中显示了视图的表示图标（）和视图的名称（导航器）。



图 2-7 导航器视图工具栏

2. 视图的两个下拉菜单

一个视图含有两个下拉菜单。在视图栏上右击鼠标，会出现一个下拉菜单如图 2-8 所示。可以通过该下拉菜单调整视图的位置、大小、显示方式等。单击视图栏上“ ”图标，显示的是视图的另一个下拉菜单，如图 2-9 所示。可以通过该下拉菜单调整视图中的文件或文件夹的排序方式、过滤掉不想显示的文件等。

2.2.4 编辑器

在工作台的透视图中通常含有一个编辑器（editor）区域、一个或多个视图。在图 2-4 显示的“工作台”窗口中，包含一个“Welcome”的编辑器和“Tasks”、“Navigator”两个视图。



图 2-8　导航器视图的调整方式

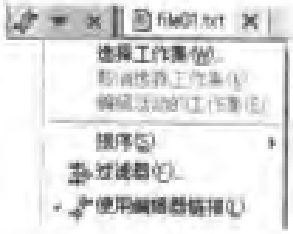


图 2-9　导航器视图功能菜单

1. 打开一个编辑器

打开一个编辑器有以下三种方法：

- 在“导航器”视图中双击要打开的文件。
- 在“导航器”视图中右击要打开的文件，在下拉菜单中选择“打开”。
- 在“导航器”视图中右击要打开的文件，在下拉菜单中选择“打开方式”，从中选择一个想要打开该文件的方式。

打开后的编辑器如图 2-10 所示，“project_lcy.txt”编辑器的顶部蓝条（project_lcy.txt）是该编辑器的标题部分，显示了该编辑器的名称为：“project_lcy.txt”，在名称的左侧是该编辑器连接文件的所属类型的图标（）。双击编辑器标题，则该编辑器会最大化，也可以通过鼠标来调整编辑器的大小。

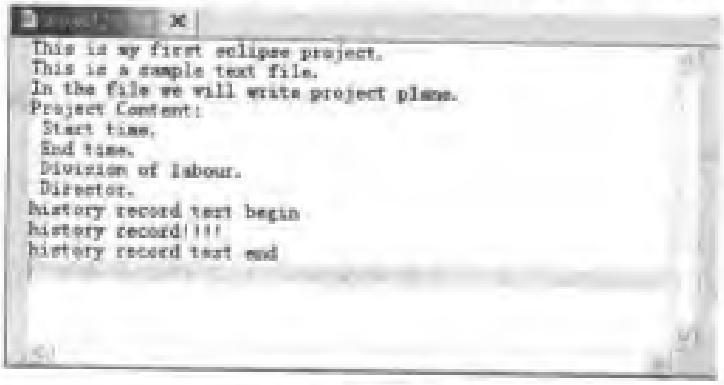


图 2-10　编辑器

2. 外部编辑器（external editor）

不同类型的文件连接不同的编辑器。当连接的编辑器中类不存在时，工作台会连接一个外部编辑器。例如，打开一个“.doc”为扩展名的文件，则工作台就会连接 Microsoft Word，使 Microsoft Word 成为该“.doc”文件的编辑器，同时工作台的菜单栏和工具栏会随之发生变化。

3. 文件的默认打开方式

一种文件的默认打开方式是可以选择的，选择菜单栏中的“文件（F）”→“首选项（P）”→“工作台”→“文件关联”，如图 2-11 所示。在该窗口中可以添加、删除文件

的种类，也可以编辑与文件类型相关联的编辑器类型。



图 2-11　设定一种文件的默认打开方式

2.2.5　工具栏

在“工作台”窗口中，含有三种工具栏（toolbar）：主工具栏（main toolbar）、视图工具栏（view toolbar）和快捷方式栏（shortcut bar）。

主工具栏又称为工作台工具栏（见图 2-4），它被显示在“工作台”窗口的顶部，在菜单栏的下面。根据当前不同的透视图，主工具栏会显示不同的内容。

视图工具栏如图 2-4 所示，对一个视图的操作会被罗列在视图工具栏中，有些操作被隐藏在下拉箭头的下拉框中。

快捷方式栏被显示在整个“工作台”窗口的左侧，是一个垂直的工具栏。通过该工具栏可以打开一个透视图，也可实现不同透视图之间的转换。

2.2.6　第一个项目

通过本节以上部分的介绍，读者应该对 Eclipse 的工作台有了一个整体的认识。下面来制作一简单的小项目，通过这个项目不但能够对上面介绍的概念有一个进一步的认识，而且能够了解工作台的一些基本操作。

1．创建一个简单项目

创建一个新的项目有三种方法：

- 在菜单栏中选择“文件（F）”→“新建（N）”→“项目（R）...”。

- 在主工具栏中单击“新建”快捷按钮（ ），在弹出的窗口中选择要创建的项目。
- 单击“新建”快捷按钮旁边的下拉标记（ ），在弹出的下拉框中选择“项目（R）...”。

在这里，采用第一种方法：

（1）“文件（F）”→“新建（N）”→“项目（R）...”，在弹出的窗口的左面窗口中选择“简单”，右面的窗口选择“项目”。

（2）单击“下一步（N）>”，在弹出的窗口中，“项目名”一项中填入项目的名称，这里我们填入“SampleProject”。“项目内容”是项目资源存储的路径，我们采用默认的路径。

（3）单击“完成”按钮，项目创建完成。

项目创建完成后，在“导航器”视图中，项目的资源就被罗列出来了，如图 2-12 所示。



图 2-12　导航器视图

2. 隐藏在视图中不想显示的文件

在项目“SampleProject”中，目前，只有一个项目的说明文件“.project”。如果想将这个文件隐藏，则在视图工具栏中选择下拉箭头，在出现的下拉菜单中选择“过滤器（F）...”。如图 2-13 所示，选择隐藏的文件的类型。现在选择“.*”，然后单击“确定”。



图 2-13　导航器过滤器

再次查看“导航器”视图，会发现“.project”文件已经看不见了。

3. 新建一个文件夹

要在一个项目中新建一个文件夹，有四种实现方法：

- 在菜单栏中选择“文件（F）”→“新建（N）”→“文件夹”。
- 在主工具栏中选择“新建”快捷按钮（），在弹出的窗口中选择“文件夹”。
- 单击“新建”快捷按钮旁边的下拉标记（），在弹出的下拉框中选择“文件夹”。
- 右击项目，在下拉菜单中选择“新建（N）”→“文件夹”。

在这里，采用最后一种方法：右击项目，在下拉菜单中选择“新建（N）”→“文件夹”。在弹出的窗口中的“文件夹名称”一项中填写文件夹的名称，现在填“FirstProject”。单击“完成（F）”按钮，则“FirstProject”文件夹创建完毕。

注意：不要在该名称的开始和结束处输入空格，不要输入特殊的字符（如“*”），而且，不要输入已经存在的文件夹的名称。如果出现这些现象，Eclipse 会发出错误提示，并屏蔽“完成（F）”按钮。

4. 新建一个文件

要在一个项目中新建一个文件夹，有四种方法：

- 在菜单栏中选择“文件（F）”→“新建（N）”→“文件”。
- 在主工具栏中选择“新建”快捷按钮（），在弹出的窗口中选择“文件”。
- 单击“新建”快捷按钮旁边的下拉标记（），在弹出的下拉框中选择“文件”。
- 右击项目，在下拉菜单中选择“新建（N）”→“文件”。

在这里，采用最后一种方法：右击项目，在下拉菜单中选择“新建（N）”→“文件”。在弹出的窗口中的“文件夹名称”一项中填写文件夹的名称，我们填“project_lcy.txt”。单击“完成（F）”按钮，则“project_lcy.txt”文件创建完毕。

注意：和新建一个文件夹相同，不能随意输入空格和特殊字符（如“*”）。

5. 编辑和保存文件内容

在“project_lcy.txt”文件中填入如下内容：

```
This is my first eclipse project.
This is a sample text file.
In the file we will write project plans.
Project Contents:
  Start time.
  End time.
  Division of labour.
  Director.
  Project.
```

这时，“project_lcy.txt”编辑器的标题栏上出现一个“*”号，这表明该文件有没有保存的内容。

保存一个文件有以下两种方法：

- 选择菜单栏中的“文件（F）”→“保存（S）”。该方法的快捷键是 Ctrl+S。
- 在主工具栏中选择“保存”按钮（）。

2.2.7　标记

标记（Markers）是连接工作台资源的一些对象。在一个工作台中可以有很多种不同种类的标记，下面对不同种类的标记描述如下。

- 书签（bookmark）——用来连接文件中特定位置的一个指示。在“书签”视图中可以找到当前所有的书签。
- 任务（task）——用来标记资源所负责的任务。在“任务”视图中，显示所有的任务。
- 问题（problem）——表示程序文件之中所存在的问题。在“任务”视图中表示问题标记。
- 错误（error）——程序文件在语法上或编译期间所出现的问题。在“任务”视图中表示错误标记。
- 警告（warning）——程序文件在语法上或编译期间所出现的需要注意的问题。在“任务”视图中表示警告标记。
- 信息任务——信息标记指示信息性任务的源位置。信息标记被看作是问题任务并可显示在“任务”视图中。

2.2.8　书签及书签视图

书签（bookmark）是用来标示经常用到的资源的导航。通过书签，能够很快地找到未完成工作的位置或常用片段的位置。当工程中含有很多的文件时，要查找个别片段是很麻烦的事情，这时如果使用书签，则就很方便了。

下面举例说明书签的创建、使用和删除。

1．书签的创建

假如，要在“project_lcy.txt”文件中的“Project Content”处建立一个书签。具体步骤如下：

（1）如图 2-14 所示，在“Project Content”的左侧，右击鼠标，在弹出的下拉菜单中，选择“添加书签（K）...”。

（2）在弹出的窗口中，填入书签的名称。这里填入“project content”。这时，可以看到在编辑器的左侧标记栏上，“Project Content”位置出现如图 2-15 的蓝色标记，它就是“书签”图标。

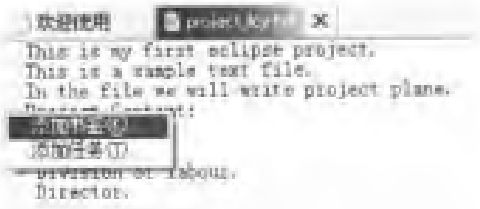


图 2-14　添加书签

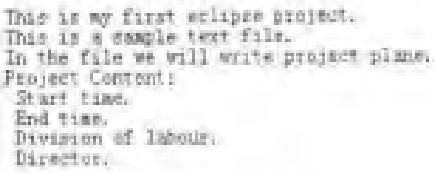


图 2-15　书签图标

2. 显示现有的书签

要想在多个文件中快速找到所需要的书签，还需要用到书签视图。选择菜单栏中“窗口（W）”→“显示视图（V）”→“书签”，如图 2-16，目前只创建了一个书签“project content”，所以，只有该书签显示出来。在书签视图中显示如下信息：

- 描述——书签名。
- 资源——书签所在的文件。
- 在文件夹中——文件所在的文件夹。
- 位置——书签所在的行数。



图 2-16　书签视图

如果想显示书签位置，则双击“书签”视图中的书签，则书签所在的那一行会被选中。

3. 删除书签

删除一个书签很简单，在编辑器的左侧标识栏中，右击想要删除的书签标记，在弹出的下拉框中选择“除去书签（K）”就可以了。双击编辑器的左侧标记栏中的“书签标记”也可以删除书签。如果打开“书签”视图，则被删除的书签也会同时从“书签”视图中消失。

在没有使用 Eclipse 前，我们标注没有完成代码的时候，会用注释方式，这样不但费时，而且易忘。在使用了书签后发现，该标记使用起来很方便而且不必在文件中写内容，从而能够保持程序的可读性和可维护性。

2.2.9　任务及任务视图

1. “任务”视图

“任务”视图中标记了工作台中各种资源所生成错误、警告、问题。比如，一个 Java 源代码文件在编译期间生成了一些警告或错误，“任务”视图会自动记录下来，并显示出来。图 2-17 显示的是一个 Java 源代码在编译期间在“任务”视图中显示出来的警告和错误。



图 2-17　任务视图

也可以用手工的方法将任务添加到任务视图中。其中有两种方式：无关联的任务和相关联的任务。

2. 无关联的任务（Unassociated tasks）

无关联的任务是指与工作台资源无关的任务。下面举例说明无关联的任务创建的过程：

（1）在“任务”视图中单击“新建任务”（）图标。

（2）在弹出的窗口中添加任务信息，如图 2-18。比如，在“描述（D）”一栏中填入“Unassociated tasks Test”。



图 2-18　新建任务

（3）单击“确定”按钮，任务就建好了。创建后的任务如图 2-19 所示。

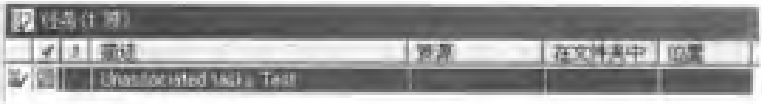


图 2-19　任务视图

3. 相关联的任务（Associated task）

相关联的任务是与工作台资源有关联的任务。下面举例说明相关联的任务创建的过

程：

（1）打开文件 project_lcy.txt。

（2）在编辑器的左侧标记栏中，右击鼠标，在下拉菜单中选择“添加任务（T）...”。

（3）图 2-20 显示的是新建任务的信息，其中“描述”一项中填入对新任务的叙述，现在填入“Associated task test”。

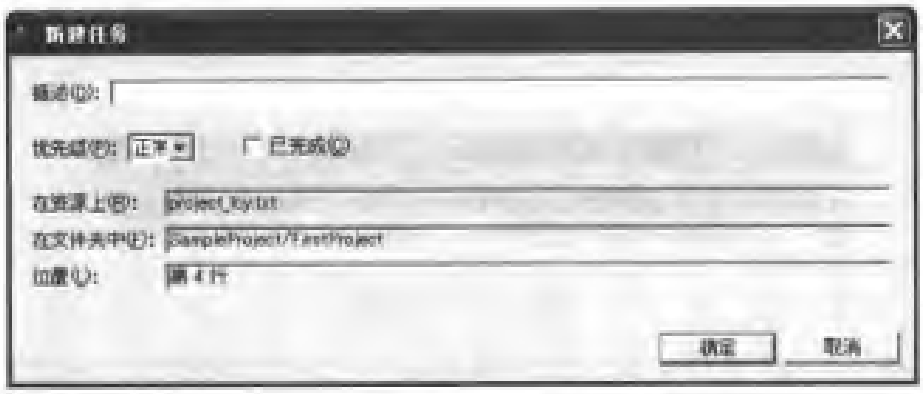


图 2-20 新建相关联任务

（4）单击“确定”按钮后，新任务创建完毕。建成后在编辑器的左侧标记栏中会出现“”标记，同时，在“任务”视图中会出现“Associated task test”任务的描述，如图 2-21 所示。



图 2-21 任务视图

4. 删除任务标记

删除一个任务标记有两种方法：

- 右击编辑器左侧标记栏中的任务（）标记，在出现的下拉菜单中选择“除去任务（T）”。
- 在“任务”视图中选中要删除的任务，单击“任务”视图标题栏中的“删除”（）按钮。

2.2.10 搜索及搜索视图

搜索是在工作台中查找一段文本字符串或文件。搜索动作是在项目开发中经常执行的，正确、灵活地运用搜索功能可使项目开发事半功倍。

1. 将“搜索”快捷图标显示在“视图工具栏”中

在安装好 Eclipse 之后，“搜索”快捷图标（）就会默认的表示出来，如果想隐藏

或显示该快捷图标，请按照以下方法控制：

（1）选择菜单栏中的“窗口（W）”→“定制透视图（Z）...”。

（2）在弹出的窗口中，选择“其他”在下拉项中找到“搜索”，如果要隐藏则去掉这一选项，如果要显示则反之。

（3）单击“确认”按钮。

2. 开始搜索

有以下几种方法打开“搜索”窗口：

（1）选择菜单栏中的“搜索（A）”，在下拉框中有三个选项：

- 搜索（A）——快捷键为 Ctrl+H，默认进入“搜索”窗口中的“文件搜索”。
- 文件搜索（F）——直接进入“搜索”窗口中的“文件搜索”。
- 帮助（H）——直接进入“搜索”窗口中的“帮助搜索”。

（2）单击“视图工具栏”中“搜索”快捷图标（）。

（3）单击“搜索”视图中“搜索”快捷图标（）。

打开后的“搜索”窗口包含四个子对话框：文件搜索、Java 搜索、帮助搜索和插件搜索，如图 2-22 所示。

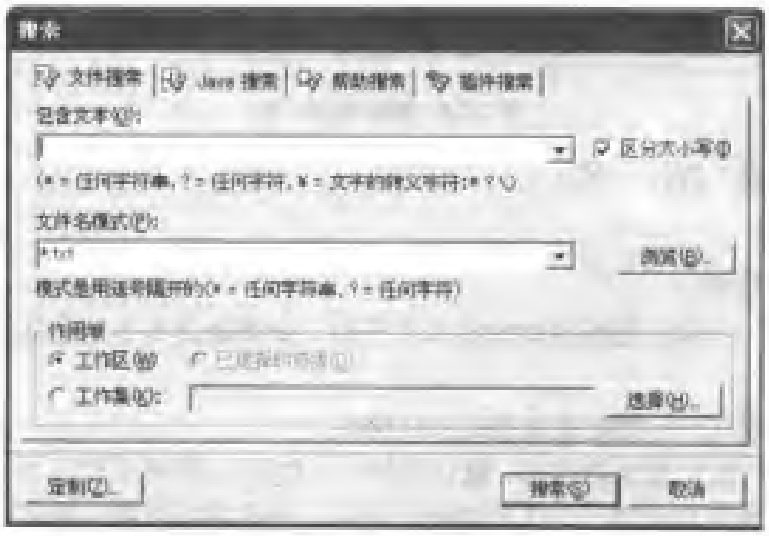


图 2-22　搜索窗口

3. “搜索”视图

图 2-23 展示了一个“搜索”视图。



图 2-23　搜索视图

在“搜索”视图标题栏中，显示了搜索的内容及在搜索范围内的搜索结果条数。如图 2-13，在标题栏中显示：“Pro”一在工作区中出现 4 次。这表明搜索的文本字符串是：“Pro”；搜索区域是：“工作区”；搜索结果是有四处位置符合搜索条件。

“搜索”视图的工具栏的各项含义如下：

- 显示下一个匹配（ ）——显示下一个符合搜索条件的结果。
- 显示前一个匹配（ ）——显示上一个符合搜索条件的结果。
- 除去所选匹配（ ）——将当前的检索结果从“搜索”视图中删除。该选项可以使搜索的范围变小，从而使搜索的目标更明确。
- 除去所有匹配（ ）——将“搜索”视图中的所有内容从“搜索”视图中删除。
- 搜索（ ）——打开“搜索”窗口。
- 上一次搜索结果（ ）——显示搜索记录。单击该项，会将近来几次的检索结果罗列出来，如图 2-24。从中可以选择你想恢复的“搜索”结果。如果单击最后一项“清除历史记录（C）”，则列表中的历史记录就会被全部删除。

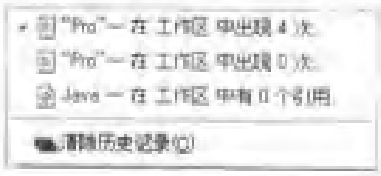


图 2-24　搜索历史记录

- 关闭“搜索”视图（ ）。

4．文件搜索

见图 2-22，其中：

- 包含文本——填入想要搜索的文本字符串。如果区分大小写，还需要选中“区分大小写”。
- 文件名的模式——限定搜索源的文件模式。
- 作用域——限定搜索的范围。默认选择“工作区”，是对工作区内已打开的项目进行搜索。也可以创建一个工作集进行搜索，该选项可以缩小搜索的范围。

下面举例做详细说明。

（1）为了使搜索的效果更明显，可先在项目“SampleProject”中添加一个“file01.txt”文件，并在其中写入如下内容并保存。

```
Project
project
Perspectives, views and editors
A window contains one or more perspectives.
A perspective consists of views and editors for working with your resources.
```

（2）按照第一种方法打开“搜索”窗口，在“文件搜索”对话框中填入以下内容：

- 包含文本——“Project”，并选中“区分大小写”；
- 文件名的模式——“*.txt”；

- 作用域——“工作区”。

（3）单击“搜索”按钮开始搜索。搜索后的结果在“搜索”透视图中表示出来，如图 2-25。从图中可以看到“Project”字符串在工作区中共搜索到 2 个：file01.txt 文件中和 project_lcy.txt 文件中。



图 2-25　搜索视图

（4）双击“搜索”透视图中的任意结果，则搜索的字符串就会被反映到“编辑器”中。比如，双击“搜索”透视图中的“file01.txt”，则“file01.txt”文件的编辑器就成为编辑器区域的当前编辑器，“file01.txt”文件中的“Project”字符串就会被选中，并在所有含有“Project”字符串的对应行的左侧标记栏中出现黄色箭头（⇨）。

2.2.11　比较

“工作台”允许用户比较多个资源，并且提供一个“比较”编辑器，用来显示比较结果。在“工作台”提供的比较中提供了三种比较方式：

- 文件相互之间的比较
- 文件和本地历史记录的比较
- 文件和版本管理服务器上资源的比较

这里只介绍前两种比较方式，最后一种比较会在第 7 章中进行详尽的介绍。

1. 文件内容之间的比较

在介绍这部分内容之前，先增加一个文件“file02.txt”，内容如下：

```
Compare test line1
Compare test line2
Compare test line3
Compare test line4
Compare test line5
```

将文件“file01.txt”的内容换成：

```
Compare test line1
Compare test line2
Compare test line3
Compare test line4
Compare test line5
```

可以看出来文件“file01.txt”和文件“file02.txt”的内容没有任何差异。下面对文件“file01.txt”和文件“file02.txt”进行比较：

（1）在“导航器”视图中，先选中文件“file01.txt”，按住“Ctrl”键，再选中文

件“file02.txt”。这样文件“file01.txt”和文件“file02.txt”就都被选中了。

（2）在文件“file01.txt”或文件“file02.txt”上右击鼠标，在下拉菜单中选择“比较（A）”→“相互（E）”。这时会弹出如图 2-26 所示的窗口。这是因为文件“file01.txt”或文件“file02.txt”的内容完全相同。



图 2-26　比较窗口

（3）改变文件“file02.txt”的内容为如下所示：

```
Compare test line1
Compare test line2
Compare test
Compare test line4
Compare test line5
Compare test line6
```

（4）执行步骤（1）、（2），会弹出“比较”编辑器，如图 2-27 所示。在“比较”编辑器的左侧是文件“file01.txt”，右侧是文件“file02.txt”。

连接左右窗格的差异线指示文件之间的区别。从图中可以看出，“file01.txt”的第三行与“file02.txt”的第三行不同，则它们由差异线连接起来。

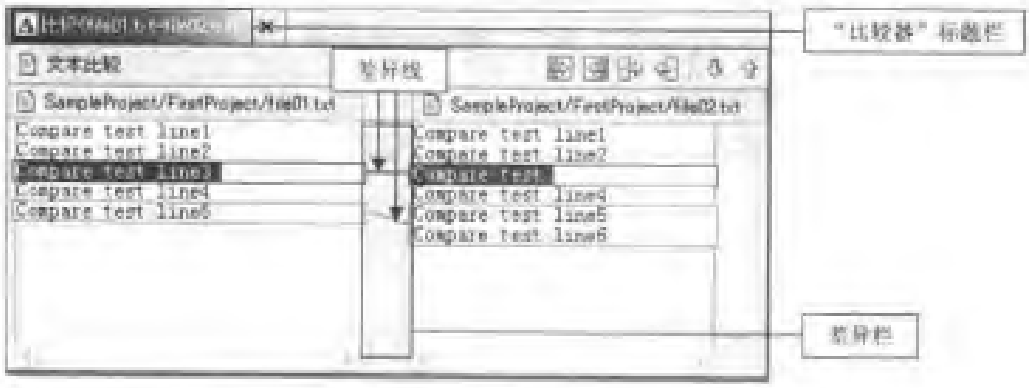


图 2-27　比较编辑器

（5）在“比较”编辑器的右上方，有一排按钮图标：

- “将所有内容从左边复制到右边”（）——将左边文件中的所有内容复制到右边文件中。
- “将所有不冲突的更改从右边复制到左边”（）——将所有不冲突的更改从右边的文件复制到左边的文件。

- “将当前更改从左边复制到右边”（）——如果单击该按钮，则文件“file01.txt”的第三行的“Compare test line3”将覆盖文件“file01.txt”的第三行的“Compare test”。
- “将当前更改从右边复制到左边”（）——如果单击该按钮，则文件“file02.txt”的第三行的“Compare test”将覆盖文件“file01.txt”的第三行的“Compare test line3”。
- “选择下一个更改”（）——当前差异线变为下一条差异线。
- “选择上一个更改”（）——当前差异线变为上一条差异线。

比如，单击“将当前更改从左边复制到右边”，则变化情况如图 2-28 所示。

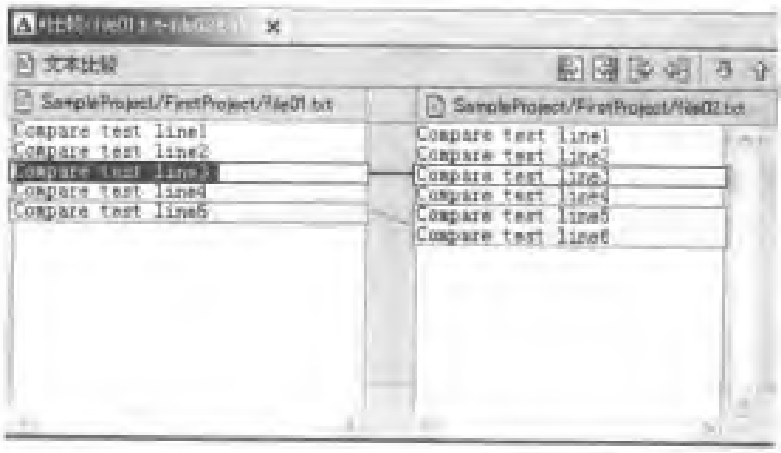


图 2-28　比较编辑器

保存刚才的变更，则两个文件第三行的差异线就消失了，当前差异线变更为下一条差异线，如图 2-29 所示。

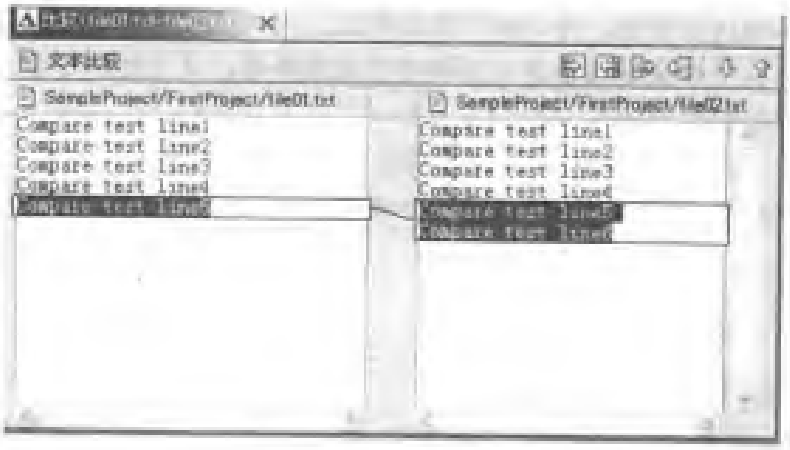


图 2-29　比较编辑器

2. 文件和本地历史记录的比较

当在“工作台”中保存可编辑文件时，“工作台”就会自动添加该文件的本地历史

记录，并记录所作的更改。可以通过与本地历史记录的比较了解文件的变更情况。

在“导航器”视图中，右击文件“file02.txt”，在下拉框中选择“比较（A）”→“本地历史记录（L）...”，则弹出“与本地历史记录比较”窗口，如图 2-30 所示。在该窗口中的上方列出了文件“file02.txt”的各个时间点的历史记录，在窗口下部的两个窗体中列出了文件“file01.txt”和文件“file02.txt”以及它们的差异。

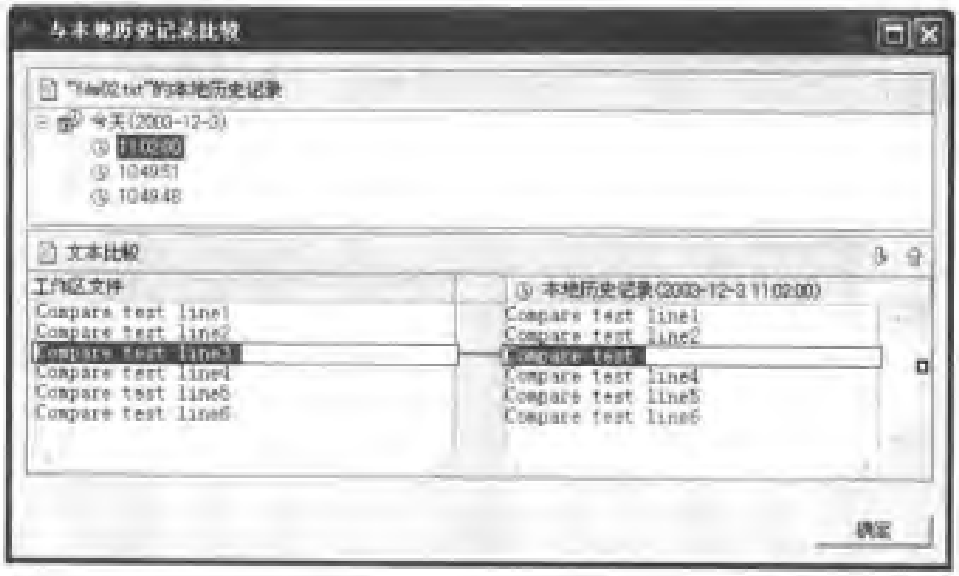


图 2-30　与本地历史记录比较窗口

2.2.12　本地历史记录

在 2.2.11 中曾介绍过，每当在“工作台”中保存可编辑文件时，“工作台”就会自动添加该文件的本地历史记录，并记录所作的更改。用户不仅可以通过这些本地历史和文件进行比较，而且可以根据需要将文件的变更重新恢复。从本地历史记录中用户可以选择需要参照或恢复的文件副本。

下面以文件“project_lcy.txt”为例说明如下。

（1）将文件“project_lcy.txt”中最后一行的“Project.”替换为如下内容：

```
history record test begin
history record!!!
history record test end
```

（2）保存文件变更。

（3）右击“导航器”视图中的“project_lcy.txt”文件，在下拉菜单中选择“替换为（T）”选项。单击“本地历史记录（L）...”，弹出如图 2-31 的窗口。这个窗口包含三个不可编辑的文本框：在窗口的上部是“project.txt”文件的本地历史记录的列表，可以在该列表中选择某个时间点的历史记录；下部的左边是当前的文件；下部的右边是在“project.txt”文件的本地历史记录的一个副本。从这个窗口中，还可以看到当前文件内

容和本地记录的不同之处。



图 2-31　当前文件与历史记录之间的替换

（4）单击“替换”按钮，进行当前文件和历史记录之间的替换。替换后的“projectlcy.txt”编辑器如图 2-32 所示。

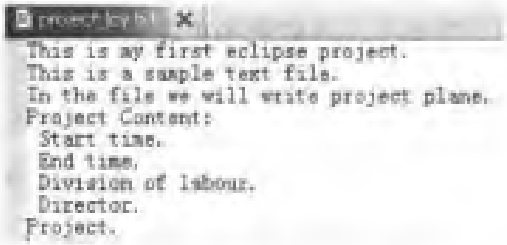


图 2-32　替换后的编辑器

2.2.13　外部工具

外部工具允许用户使用“工作台”配置和运行程序、批处理文件、Ant 构建文件和其它对象。使用外部工具能够有效的构建项目，是项目管理的得力助手。Eclipse 的外部工具框架处理两大类外部工具：

- Ant 构建文件
- 其他构建文件

Ant 构建文件在第 6 章“Eclipse 环境下的 Ant 插件的应用”中介绍，在本节中只介绍非 Ant 外部工具。

下面举一个小例子做详细：

（1）在 D 盘下建一个 Windows 的批处理文件 copy_document.bat。输入：copy "D:\document.txt" "D:\eclipse"。其含义是将 D 盘下的“document.txt”文件复制到 D 盘下的“eclipse”文件夹下。

（2）在“导航器”视图中选中“SampleProject”项目，选择菜单栏中的“运行（R）”→“外部工具（E）”→“外部工具（E）...”。

（3）弹出如图 2-33 所示的窗口，在窗口的左侧选择“程序”，单击“新建”按钮。

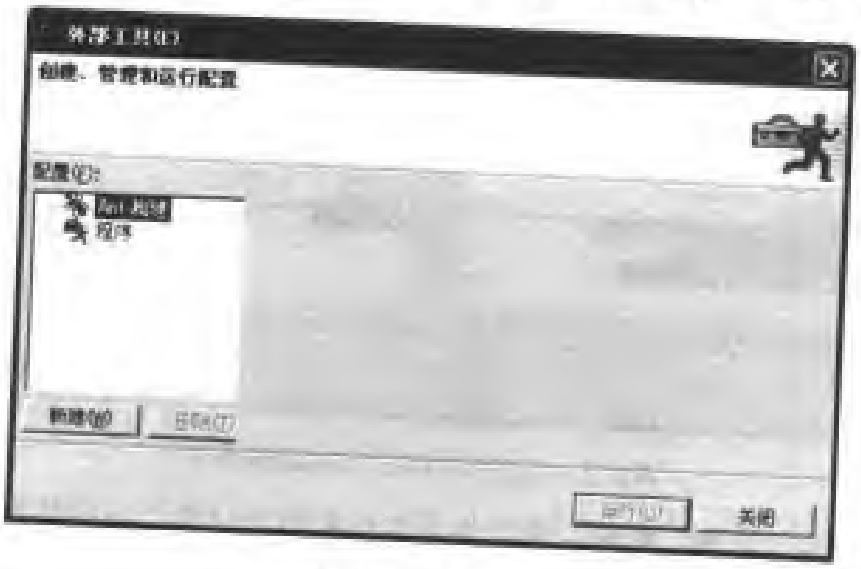


图 2-33 外部工具窗口

（4）在弹出的窗口中填写外部工具配置的内容，如图 2-34 所示。

（5）单击“应用（Y）”按钮。

（6）单击“运行（U）”按钮，在“控制台”中将会输出执行的信息，本例的信息如图 2-35 所示。查看 D 盘下的“eclipse”文件夹会发现，“document.txt”文件出现了。

2.2.14 更新管理器

Eclipse 更新管理器可用来以可预知的方式呈示和浏览 Eclipse 更新站点。更新站点可包含显示在树视图中的类别，这些类别又包含一些功能部件。可以在更新管理器内的“预览”视图中检查功能部件。如果功能部件是有效的安装候选功能部件，则它将包含一个将启动 Eclipse 安装向导的按钮“立即安装”。完成安装任务后，该功能部件将成为产品的一部分。

下面介绍如何使用更新管理器来更新功能部件。

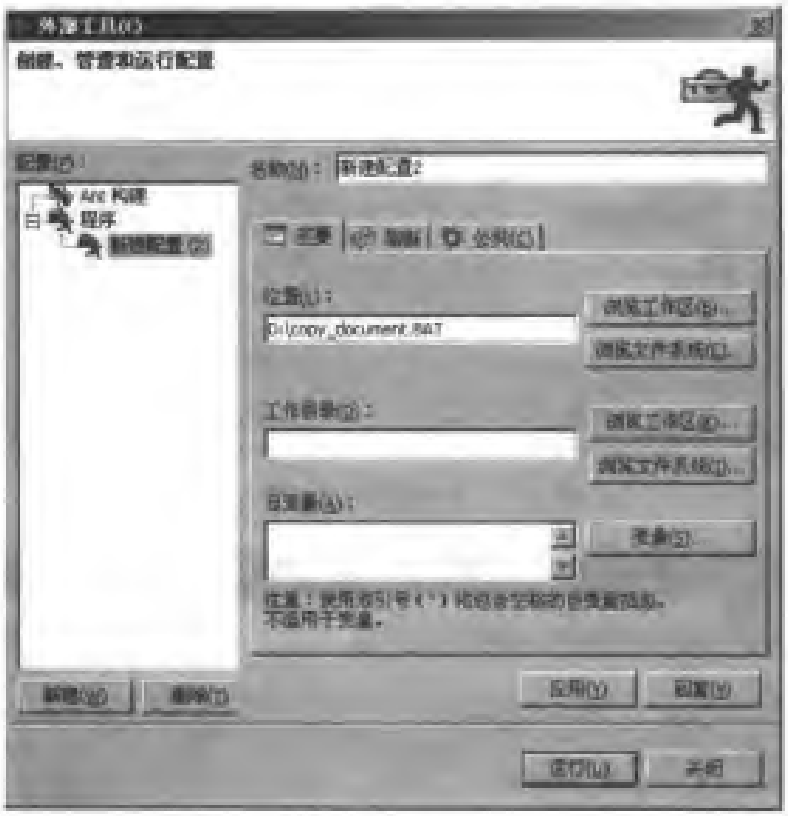


图 2-34　外部工具窗口



图 2-35　控制台

1．激活更新管理器的更新功能

更新管理器能够运行可接受来自 Web 页面的请求的 servlet，但是此功能不是自动的，必须在“首选项”→“安装 / 更新”→“Web 触发的更新”页面中激活它，如图 2-36 所示。当单击“确定”按钮时，更新管理器将激活可处理入局请求的应用程序服务器。

2．对现有功能部件进行更新（需要访问因特网）

（1）单击“帮助”→“软件更新”→“新的更新”。这将跟与产品的功能部件相关联的 Web 站点联系，以发现那些功能部件的哪些版本可用。“可用的更新”页面中提供了潜在的升级，如图 2-37 所示。

（2）选择想要升级的功能部件版本，并单击“下一步”。

（3）仔细复查已升级的功能部件的许可证协议。如果所有这些许可证的条款都可以接受，则选择“我接受许可证协议中的条款”。如果许可证条款不可接受，则不要继续下载功能部件。



图 3-36　更新管理器首选项窗口

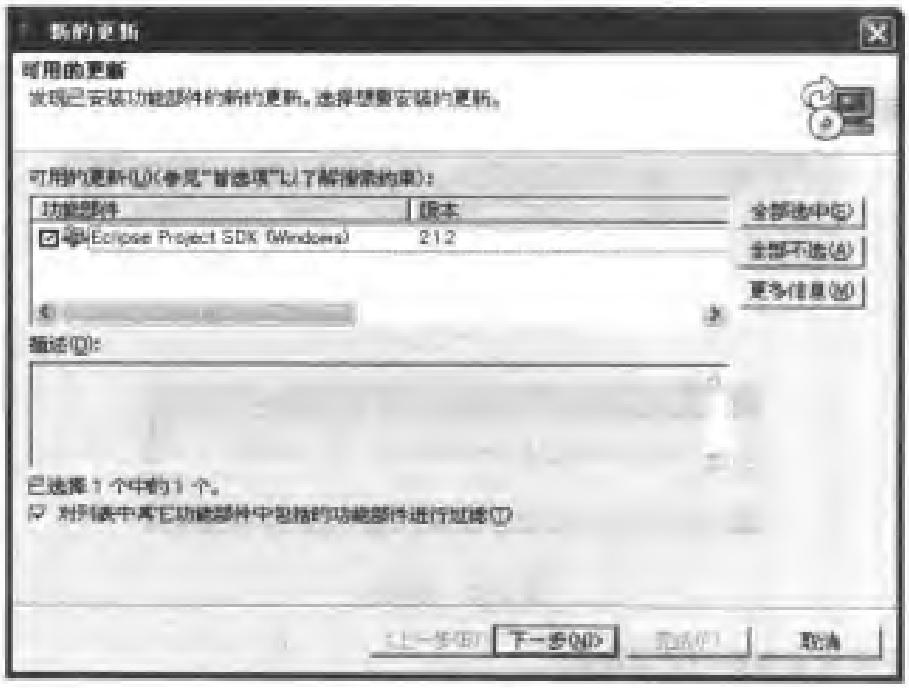


图 2-37　可用更新窗口

（4）功能部件版本可以由提供它们的公司以数字形式签名。这使用户能够更容易验证要下载并安装的功能部件和插件是否都来自可信赖的供应商。

警告：由于插件可能具有危害甚至是怀有恶意的，所以只应从值得信任的组织中下载功能部件。如图 2-38 所示的功能部件验证，警告此功能部件未加数字符号。如果确认功能部件是正确的，就可单击“安装”继续进行下载和安装。

（5）一旦成功地下载了所有功能部件和插件，并将它们的文件安装到本地计算机上的产品中，就将阐明组合了这些功能部件和插件的新配置。当询问你是否退出时，单击“是”，并重新启动“工作台”以使更改生效。



图 2-38　功能部件验证

3. 安装/更新透视图

打开安装/更新透视图有两种方式：

- 单击“帮助”→“软件更新”→“更新管理器”。
- 打开“窗口”→“打开透视图”→选择“安装/更新透视图”。

工作台会显示出如图 2-39 所示的更新管理器视图画面，该视图包括以下四个相关视图，它们分别是：

（1）“安装配置”视图。“安装配置”视图（见图 2-39 左上角）允许用户管理工作区配置。“当前配置”文件夹显示哪些本地安装位置是工作区配置的一部分，以及在那些位置中安装了哪些功能部件。“配置历史记录”文件夹显示对工作区配置所作更改的历史记录。“已保存配置”文件夹显示已保存的所有配置（通常表示可用作要恢复至的基础的固定配置）。

（2）“功能部件更新”视图。“功能部件更新”视图（见图 2-39 左下角）允许用户定位和安装新功能部件和功能部件更新。该视图包含三个标准文件夹。“要访问的站点”文件夹包含正在使用的产品预先配置的更新站点。“可用的更新”是用于搜索更新站点的内置搜索结果文件夹。“我的电脑”文件夹是一个文件资源管理器，可用于文件系统中的更新。

除标准文件夹之外，可通过添加更新站点书签，在“功能部件更新”视图中创建附加内容和定制搜索。



图 2-39　安装/更新面面

（3）“预览”视图。“预览”视图（见图 2-39 右半部分）显示有关当前活动选择的详细信息。当选择更改时，其内容也跟着更改。在某些情况下，“预览”只是选择的扩展描述，或者是根据选择提供的建议操作的页面。在其他情况下，可以使用预览来针对选择执行特定的操作。例如，当前配置的预览显示有关为工作区定义的基本功能部件的信息。所选历史记录的预览显示有关历史记录条目的相应配置信息。如果可以针对该选择执行某个操作，则在预览中显示为操作按钮。

（4）“外部预览”视图。“外部预览”视图是用来显示 html 信息的嵌入式浏览器，此视图仅在 Windows 系统上可用。

一般情况下，当遵循链接访问 html 页面时才会使用“外部预览”。

2.3　资源管理

2.3.1　资源与工作台

1. 什么是资源

资源是存在于工作台中的项目、文件夹和文件的统称。“导航器”视图提供了资源的分层视图并允许用户打开它们以进行编辑。其他工具可以不同的方式显示和处理这些资源。工作台中有以下三种基本资源类型：

（1）项目。项目用于构建、版本管理、共享和资源组织。与文件夹相似，项目映射

到文件系统中的目录。项目包含文件夹和文件。（当创建某个项目时，将在文件系统中为它指定位置。）

项目的状态为打开或关闭。关闭项目后，就不能在工作台中更改它。关闭的项目的资源将不会出现在工作台中，但这些资源仍会驻留在本地文件系统上，关闭的项目需要较少的内存。由于在构建期间不会检查这些项目，所以关闭项目可以缩短构建时间。当项目处于打开状态时，可以更改该项目的结构，而且用户可看到有关内容。

插入到工作台中的不同工具使用它们自己专用的项目、文件夹和文件类型。

（2）文件夹。与文件系统中的目录类似。在工作台中，文件夹包含在项目或其他文件夹中。文件夹可包含文件和其他文件夹。

（3）文件。与用户在文件系统中看到的文件类似。文件的内容与平台无关。

2. 资源的层次结构

资源是按层次结构存储和显示在工作台中的。下面描述的是引用以分层结构存储和显示的资源时所使用的术语。

- 根——工作台内容的最上面一层（在文件系统中）。
- 父资源——包含另一资源的任何资源。只有项目和文件夹才可称为父资源。
- 子资源——包含在另一资源中的任何资源。只有文件和文件夹才可为子资源。

资源层次结构显示在“导航器”视图中，该视图是“资源”透视图中的其中一个默认视图。

3. 资源的树结构

图 2-40 所示的树结构（在工作台导航器视图中显示）说明了工作区中的资源的典型层次结构。树的（隐含）根目录是工作区根目录。项目是工作区根目录的直接子代。每个节点（不同于根目录）是三种资源中的其中一种，并且每个节点的名称都与它的兄弟节点的名称不同。

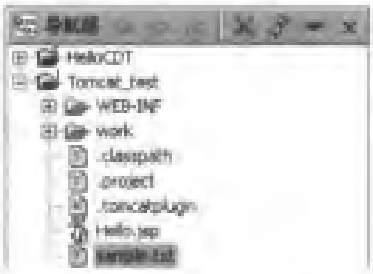


图 2-40　资源的树结构

资源的名字是任意符合命名规则的字符串。平台本身不限定资源名，它也不指定任何具有特殊意义的名称。（一种例外的情况是不能将项目命名为“.metadata”，因为此名称是内部使用的。）

4. 链接的资源

项目下面的文件夹和文件可以链接至文件系统中项目位置之外的位置。这些特殊的文件夹和文件称为链接的资源。链接的资源就是存储在文件系统中项目位置之外的文件

和文件夹。这些特殊资源必须具有作为它们的父资源的项目。链接的文件夹和文件可用来将资源（出于某种原因，这些资源必须存储在用户的项目之外的某一位置）添加至用户的项目。例如，链接的文件夹可以用来存储构建输出（与源文件分开存储）。甚至还可以使用链接的资源来覆盖工作区中的其他资源，以便一个项目中的资源可以出现在另一个项目中。如果不想在工作区中出现覆盖资源的情况，则进行此操作时要慎重。一定要注意，这意味着更改一个位置中的资源将导致在重复资源中同时发生更改，删除一个重复资源将会同时删除两个资源！

处理链接的资源有一些特殊的规则。因为它们必须就处于项目下面，所以不能将链接的资源复制或移动到其他文件夹中。删除链接的资源不会导致删除文件系统中的相应资源，但是，删除链接文件夹的子资源将导致从文件系统中删除它们。

在 Eclipse 平台的基础上构建的一些插件与链接的资源不兼容。如果情况是这样，则可以完全禁用链接的资源功能部件以防止在工作区中创建它们。可以从“窗口（W）”→“首选项（P）”→“工作台”→“链接的资源首选项”页面禁用链接的资源。某些类型的项目或小组资源库提供程序还可能不允许在一些项目中创建链接的资源。

2.3.2 对资源进行操作

1. 创建资源

（1）创建项目。如果用户想要创建自己的项目，可以按照如下的步骤进行：

① 在主菜单栏上，单击“文件（F）”→“新建项目（N）”→“项目（R）”。“新建项目”向导打开。

② 从左列中选择类别，然后从右列中选择要创建的项目类型。单击“下一步”。

③ 在“项目名（P）”字段中，输入新项目的名称。

④ 用户创建的项目将映射至文件系统中的目录结构。默认文件系统位置显示在“位置”字段中。如果想要在另一位置创建项目及其包含的资源，则清除“使用缺省值（D）”复选框，并指定新位置。

⑤ 如果想要新项目依赖于一个或多个其他项目，则单击“下一步”，并选择要引用的项目。

⑥ 单击“完成”。新项目就创建成功了，并会被显示在“导航器”视图中。

（2）创建文件夹。如果用户想要创建新的文件夹，方法如下：

① 在“导航器”视图中，右击想要在其中创建新文件夹的项目或文件夹。

② 从弹出菜单中，选择“新建（W）”→“文件夹”。

③ 输入新文件夹的名称，并单击“完成”。

（3）创建文件。如果用户想要创建新的文件，方法如下：

① 在“导航器”视图中，右击想要在其中创建新文件的项目或文件夹。

② 从弹出菜单中，选择“新建（W）”→“文件”。

③ 指定文件的名称，包括文件扩展名（例如，file.txt）。

④ 单击“完成”。

（4）创建链接的资源。如果用户想要创建链接至该项目位置之外的链接资源，则应首先创建链接的文件夹，其步骤为：

① 在“导航器”视图中，右击想要在其中创建新文件夹的项目。

② 从弹出菜单中，选择“新建（W）”→“文件夹”。

③ 指定文件夹的名称，因为它将出现在工作台中。此名称可与文件系统中文件夹的名称不同。但是不能和项目中已有的文件夹相同。

④ 单击“高级（A）”。

⑤ 选择链接至文件系统中的文件夹。

⑥ 输入文件系统路径，或单击“浏览（W）”来选择文件系统中的文件夹。

⑦ 单击“完成（F）”。

要创建链接的文件，除了在“导航器”的上下文菜单中选择“新建（W）”→“文件”，而不是选择“新建（W）→“文件夹”之外，其余的步骤和上面相同。

上面的第⑥步中还可以相对于变量指定链接的资源位置，这使得更容易与其他小组成员共享包含链接的资源的项目，因为它避免使用在各个机器之间可能会有不同的硬编码的绝对文件系统路径。

如果要定义相对于路径变量的链接的资源，则应将上述步骤⑥、⑦替换为以下操作：

⑥ 单击“变量（V）”按钮。

⑦ 在获得的对话框中，选择现有路径变量或创建新的路径变量。

⑧ 如果所选变量定义了链接的资源的精确路径，则单击“确定”。否则，单击“扩展（E）”以在由路径变量描述的位置下指定文件或文件夹，然后单击“确定”。如果创建的是链接文件夹，那么最终指定的链接目标应该是文件系统中的文件夹。如果创建的是链接文件，那么最终指定的链接目标应该是文件系统中的文件。

⑨ 单击“完成（F）”。

一旦创建了链接的资源，将不能更改在上面的步骤⑥或⑧中输入的链接目标路径。用户要想变更链接资源，只能重新建立新的链接资源。

2. 查看资源属性

要显示工作台资源的各种属性，则：

（1）右击“导航器”视图中的资源。

（2）从弹出菜单中选择“属性（P）”。

（3）显示的属性类型取决于已选择的特定资源和安装到工作台中的功能部件和插件。

3. 移动资源

可以将资源从一个工作台位置移至另一个位置（例如，从一个项目移至另一个项目），方法为：

（1）在“导航器”视图中，选择想要移动的资源。

（2）右击该资源，从弹出菜单中选择“移动（V）”。

（3）在“文件夹选择”窗口中（见图 2-41），选择想要将资源移动至的项目或文件夹，并单击“确定”。



图 2-41　文件夹选择窗口

还可以通过在“导航器”视图中使用鼠标将资源从原始位置拖动至新位置来移动资源。

移动链接的资源总是移动链接而不是移动它链接到的资源。例如，当移动链接的文件夹时，不会在文件系统上移动文件夹内容。记住，只能将链接的资源移动至项目而不能移动至文件夹。

4．复制资源

可以将资源从一个工作台位置复制到另一个位置（例如，从一个项目复制到另一个项目），方法为：

（1）在“导航器”视图中，选择想要复制的资源。

（2）右击该资源，从弹出菜单中，选择“复制（C）”。

（3）在“导航器”视图中，选择想要将资源复制至的项目或文件夹。

（4）从弹出菜单中选择“粘贴（P）”。

还可以通过按住 Ctrl 键并在“导航器”视图中将资源从原始位置拖动至新位置来复制资源。

复制链接的资源总是复制链接而不是复制它链接到的资源。例如，当复制链接的文件夹时，不会在文件系统上复制文件夹内容。相反，将创建链接至文件系统上的相同位置的新链接的文件夹。记住，只能将链接的资源复制至项目而不能复制至文件夹。

5．重命名资源

可以通过使用“导航器”视图中的上下文菜单上的“重命名”命令来重命名工作台资源，方法为：

（1）在“导航器”视图中，右击想要重命名的资源。

（2）从弹出菜单中，选择“重命名（M）”。

（3）输入资源的新名称。

6. 删除资源

删除资源的方法如下所述。

（1）在“导航器”视图中选择要删除的资源，删除资源有如下三种方法：

- 右击准备要删除的资源，从该项目的弹出菜单中选择“删除（D）”。
- 选中准备要删除的资源，按 DEL 键。
- 选中准备要删除的资源，从下拉菜单中选择“编辑（E）”→“删除（D）”。

（2）确认删除。如果删除的是项目，那么用户还要选择想要执行的删除类型，有两种删除类型：

- 从文件系统中删除有关项目的所有内容。
- 从工作区删除项目但保留它在文件系统中的内容。这也是默认情况。

2.3.3　浏览和查找资源

1. 在工作台中浏览资源

“导航器”视图是一个重要的视图，它显示了关于工作台的内容和层次结构中各资源如何相互关联的信息。如图 2-42 所示。在工作台中，所有资源都驻留在项目中。项目中可以包含文件夹和单个的文件。



图 2-42　导航器视图

（1）打开导航器中的资源。在“导航器”视图中，选中要编辑的文件 File.txt，可选择下列其中一个方法打开该文件的编辑器：

- 在“导航器”视图中，双击该资源，可以使用该类资源的缺省编辑器来编辑该资源，或者右击该资源并从它的弹出菜单中单击“打开（O）”。
- 如果想要使用特定编辑器来编辑资源，则应首先在“导航器”中选择该资源，然后右击该资源从弹出菜单中选择“打开方式（H）”选项，从其中选择你所想选择的编辑器来编辑该资源，见图 2-43。



图 2-43　文件打开方式

工作台将记住用户用来编辑特定文件的上一个编辑器。这使得可较为容易地一直使用同一个编辑器。可以使用“窗口（W）”→“首选项（P）”→“工作台”→“文件关联”来配置缺省时的默认编辑器。

（2）转至。“导航器”视图的“转至”操作很容易转至“导航器”中的特定资源。

① 选择“导航器”视图。

② 从下拉菜单中选择“浏览（N）”→“转至（G）”→“资源（R）”，弹出如图 2-44 所示的“转至资源”对话框。

③ 在该对话框顶部的模式字段中输入“Fil”。当用户输入文件名时，对话框会根据用户到目前为止已经输入的内容来过滤可能存在的匹配集合。

④ 从“匹配资源（M）”字段中选择 File.txt，并单击“确定”按钮。

⑤“导航器”视图将选择文件 File.txt，然后可以使用“打开导航器中的资源”中介绍的方法打开该资源。

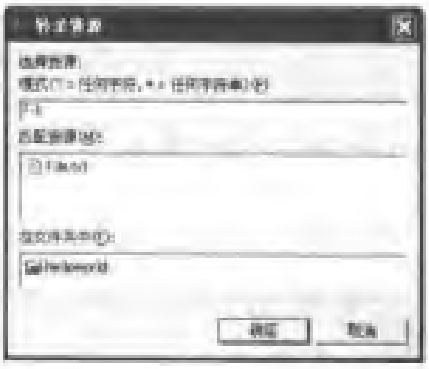


图 2-44 转至资源对话框

（3）进入。“导航器”视图显示工作台中的所有资源。可以选择项目或文件夹，并隐藏所有其他资源。下面试着对 helloWorld 项目执行此操作。

① 选择“导航器”视图。

② 右击 helloWorld 项目，从弹出菜单中选择“进入（I）”。

③ 此时，“导航器”仅显示 helloWorld 项目的内容。“导航器”视图的标题显示用户当前查看的资源的名称，如图 2-45 所示。

④ 可以使用“上一步”、“下一步”和“向上”（ ）按钮来在资源和项目 helloWorld 之间移动。



图 2-45 导航器视图查看资源工具栏

2. 快速查找资源

如果用户想快速在“导航器”视图中找到某个资源，用户可以通过转至方法来实现。如果想在编辑器中打开某个资源，而不是在视图中选定它，用户可以通过在下拉菜单中选择：“浏览（N）”→“打开资源（U）”。打开的是和转至操作相同的窗体，但是，它是将所选择的资源直接在编辑器中打开，如图 2-46 所示。

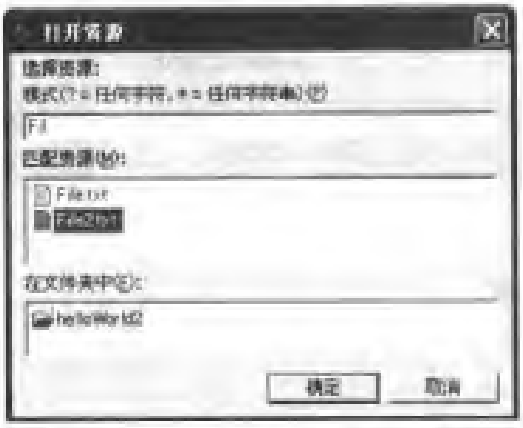


图 2-46　打开资源

3. 搜索文件

转至操作允许用户在工作台中快速地查找资源，用户还可以对工作台中的资源进行更复杂的查找。比如，要查找所有扩展名为.txt 的文件。搜索文件的步骤如下：

（1）在主工具栏上，单击搜索按钮 。或者，在下拉菜单“搜索（A）”中选择→“搜索（A）”。

（2）将*.txt 输入到“文件名模式（P）”字段并保持“包含文本（O）”字段为空白，如图 2-47 所示。



图 2-47　文件搜索窗口

（3）完成输入搜索选项（例如，将搜索限定于指定的工作集）并单击“搜索（S）”。

（4）“搜索”视图显示搜索的结果。在“搜索”视图中右击任何项可打开弹出菜单，允许用户从列表除去项、将搜索结果复制到剪贴板或重新运行搜索。要打开列示的文件之一，双击它或从右击它的弹出菜单中选择“转至文件（G）”，如图 2-48 所示。



图 2-48　搜索结果视图

4. 在文件内搜索文本

用户还可以对包含在工作台中的文件中的信息进行搜索。查找包含特定字符串的所有文件的步骤如下：

（1）在主工具栏上，单击搜索按钮 。在下拉菜单中选择“搜索（A）”→“搜索（A）”。

（2）在“包含文本（O）”字段中输入搜索字符串，或使用下拉列表来选择先前输入的搜索表达式。对于包含特殊字符“*？\”的搜索字符串，使用“\”作为转义字符（例如，想搜索在文本中包括“*.*”的文件，那么在“包含文本（O）”字段中应该填写：*.*），如图 2-49 所示。

（3）完成输入搜索选项（例如，将搜索限定于指定的文件类型），单击“搜索”。

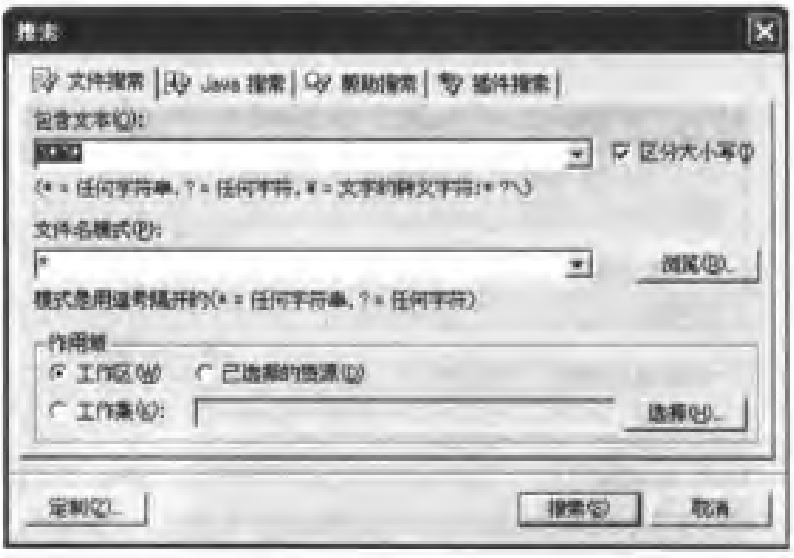


图 2-49　文件搜索窗口

（4）“搜索”视图显示搜索的结果。在“搜索”视图中，右击任何项可打开弹出菜单，允许用户从列表除去项、将搜索结果复制到剪贴板或重新运行搜索。要打开列示的

文件之一，双击它或从它的弹出菜单中选择“转至文件”。

5. 在“导航器”视图中对资源进行排序

要将“导航器”视图中的工作台资源按名称或者按文件类型排序：

（1）在“导航器”视图的工具栏上，单击菜单按钮 ▾ 以打开显示选项的下拉菜单。

（2）选择“排序（S）”。

（3）选择用户所希望的排序方式选项，包括按名称排序或者按类型排序。

6. 在“导航器”视图中显示或隐藏资源

可以选择在“导航器”视图中隐藏系统文件或生成的类文件。（系统文件就是只有文件扩展名而没有文件名的那些文件，例如.porject。）

（1）在“导航器”视图的工具栏上，单击菜单按钮 ▾ 以打开显示选项的下拉菜单。

（2）选择“过滤器（F）”。

（3）在打开的对话框中，选择想要隐藏的文件类型的复选框。

另外，可以将显示的文件限制于一个工作集：

（1）在“导航器”视图的工具栏上，单击菜单按钮 ▾ 以打开显示选项的下拉菜单。

（2）选择“选择工作集（W）”。

（3）从列表中选择现有工作集或通过选择“新建（N）”创建新的工作集。

2.3.4　导入/导出资源

2.3.4.1　导入资源

导入资源可以有两种方式：拖放或者复制和粘贴（在 Windows 操作系统中）；使用导入向导完成导入。

1. 拖放或者复制和粘贴

（1）打开操作系统的文件系统资源管理器（比如 Windows 的“我的电脑”）。

（2）选择要导入的资源，并将它拖到工作台中的“导航器”视图中的特定位置，并松开鼠标按钮，资源将被放置在指定位置。

（3）还可以通过在文件系统资源管理器中复制文件，然后在“导航器”视图中指定要导入的目标，选择“编辑（E）”→“粘贴（P）”来实现这一操作。

2. 使用导入向导完成导入

（1）右击准备要导入资源的项目。

（2）在弹出菜单中选择“导入（I）”。

（3）在“导入”向导中，选择文件系统，然后单击“下一步”，参见图 2-50。

（4）在“从目录（Y）”这一项中，输入资源的目录，或进行“浏览（R）”以选择包含导入资源的目录。

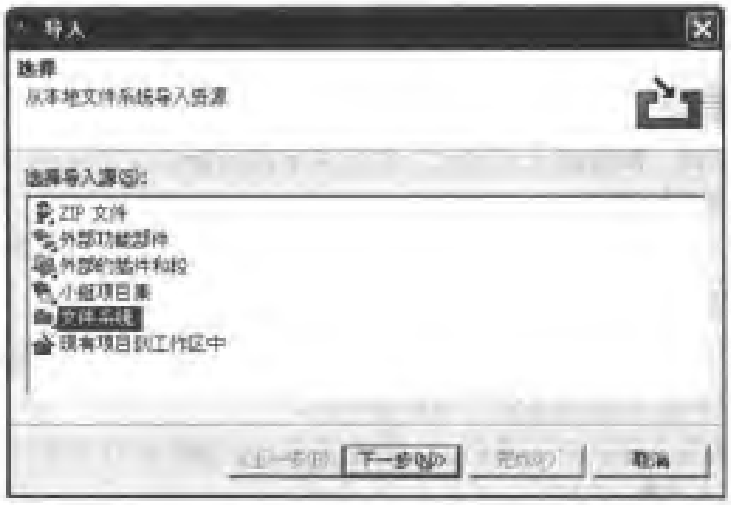


图 2-50　导入资源窗口

（5）在右窗格中，选择要导入的文件，在左窗格中选择文件夹会将它的整个内容导入工作台。变灰的复选框（如图 2-51 所示）表示在文件夹中只有一部分文件将导入工作台。可以使用“过滤类型（T）”按钮来过滤想要导入的文件的类型。



图 2-51　从本地文件系统导入资源

（6）应该已经在“至文件夹（L）”字段中填充了项目的名称。

（7）可以通过单击“浏览（W）”来更改目标项目或文件夹，选择其他的项目。在弹出的对话框图 2-52 中，选择项目“helloworld”。



图 2-52　导入至文件夹

（8）在图 2-51 的所示“选项”区域中，可以选择：覆盖现有的资源而不发出警告（O）、创建完整的文件夹结构（C）和只创建选择的文件夹（E）。

（9）完成上述步骤后单击“完成（F）”。

现在，想要导入的文件就显示在“导航器”中的对应的项目中了。

2.3.4.2　导出资源

导出资源可以有两种方式：拖放，或者复制并粘贴（在 Windows 操作系统中）；使用导出向导完成导出。

1．拖放或者复制并粘贴

（1）打开操作系统的文件系统资源管理器（比如 Windows 的“我的电脑”）。

（2）将想要导出文件从“导航器”视图拖到文件系统资源管理器中。

（3）如果用户发现导出的文件在导出的同时被从工作区中除去了，那么，你可以在拖动的同时按住 Ctrl 键，这样就可以确保复制了该文件而没有从工作区中除去该文件。

（4）还可以通过在“导航器”视图中选择文件并选择“编辑（E）”→“复制（C）”，然后，在文件系统资源管理器中粘贴该文件来实现导出。

2．使用导出向导完成导出

（1）右击准备要导出资源的项目。

（2）在弹出菜单中选择“导出（X）”。

（3）在“导出”向导中，选择文件系统，然后单击“下一步”，如图 2-53 所示。

（4）展开项目如图 2-54 所示，单击要导出资源所在的文件夹。在右窗格中，应确保要导出的资源处于选中的状态。注意，左窗格中的文件夹和项目现在有一个变灰的复选框，表示将导出它们的一些内容，但不是全部内容。可以使用选择类型按钮来过滤想要导出的资源的类型。如果预先知道只想导出某一个文件（比如 File.txt），则可以就在“导航器”视图中选择该文件而后选择“文件（F）”→“导出（X）”。“导出”向导会自动确保它就是选择要导出的惟一文件。

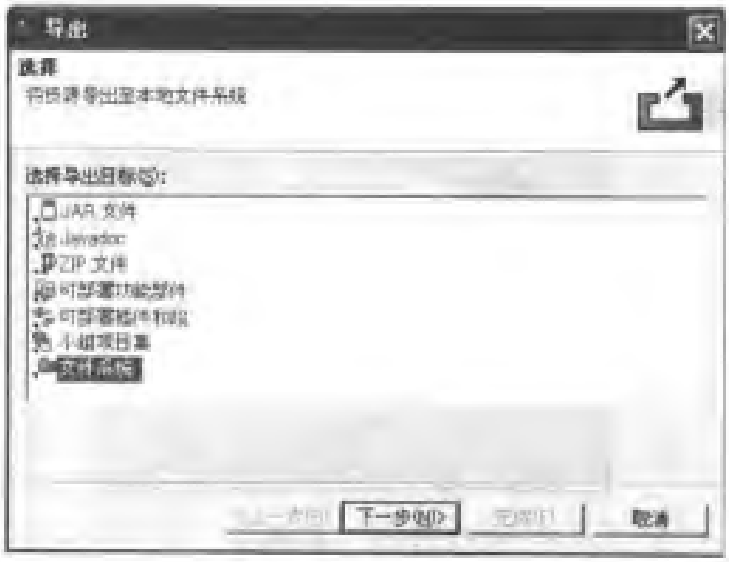


图 2-53　导出窗口

（5）在“至目录（D）”字段中，输入或进行浏览以选择文件系统中用户想要所导出资源保存的位置。如果输入了不存在的目录名，则当用户按“完成”时，“导出”向导就会为用户创建该目录名。



图 2-54　将资源导出至本地文件系统

（6）在“选项”区域中，可以选择：覆盖现有的资源而不发出警告（O）、创建完整的文件夹结构（C）和只创建选择的文件夹（E）。

完成上述步骤后单击“完成（F）”，文件将导出到指定的文件夹中。

2.3.5　插件开发中的资源

资源插件（org.eclipse.core.resources）是 Eclipse 平台中提供的基本插件之一，它帮助用户访问想要使用的文件。

2.3.5.1　资源和本地文件系统

当平台核心正在运行并且资源插件是活动的时候，工作区是由接口 IWorkspace 的实例表示的，它提供用于访问它包含的资源的协议。接口 IWorkspace 的实例表示本地文件系统中相关联的文件和目录的集合。可以从 org.eclipse.core.resources 中的资源插件类访问工作区，访问的方法：

```
ResourcesPlugin.getWorkspace();
```

当资源插件未运行时，工作区单独存在于本地文件系统中，用户可以通过标准的基于文件的工具来查看或处理工作区。在说明资源插件 API 之前先看一下磁盘上工作区的外观。

1．本地磁盘上的树状结构

安装 Eclipse 时，已将文件解压缩到用户所选择的本地磁盘的目录中，此目录称为平台根目录。此目录包含 plugins 目录以及其他目录。在平台根目录中，有一个 workspace 目录，该目录用来保存由平台创建和处理的资源。如果查看 workspace 目录，将会看到工作区中存在的每个项目的不同子目录。在这些子目录中是每个项目包含的文件夹和文件。

假如说用户的 Eclipse 安装在 D:\eclipse 中，则在 D:\eclipse\workspace 目录中，可以找到根据工作区的项目命名的子目录。这些目录称为项目的内容目录。内容目录是用户创建项目时由平台创建的。在项目的每个目录中，都会找到文件和文件夹，其布局与它们在工作区的“导航器”视图中所看到的资源树中的布局完全相同。所有文件名也都是一样的，并且无论从文件系统还是从工作区来访问，文件的内容都是相同的。

平台具有特殊.metadata 目录，用来保存平台内部信息。工作区的.metadata 目录被认为是一个“黑匣子”。有关工作区结构的重要信息（例如，项目的引用或者资源的属性）都存储在工作区的元数据部分，并且应当只能由工具通过平台 API 来访问。决不能使用一般文件系统 API 来编辑或处理这些文件。用户创建的任何资源都不可以命名为.metadata。

此外，每个项目有其自己的.project 文件，在其中保留关于项目的元数据。除了.metadata 目录和.project 文件之外，其他工具也可以使用工作区目录中的文件夹和文件。文件和文件夹可以由非集成工具来处理，例如，文本编辑器和文件系统实用程序。惟一的问题就是当用户在工作台内部和外部编辑这些文件时都必须小心（这与用户使用两个无关的独立工具来编辑文件没有区别）。工作台提供了刷新操作来协调资源的工作区视图与文件系统的实际状态。

2. 用代码编写的样本树

资源 API 允许用代码处理本地磁盘上的资源树。下面，先考察一些代码片段，以便快速地了解资源 API。资源 API 是在 org.eclipse.core.resources 中的一系列接口中定义的。以下是其中一些资源类型的接口，例如，Iproject，IFolder 和 IFile。扩展公共协议是在 IResource 中定义的。我们还要利用 org.eclipse.core.runtime 接口 IPath，它是用来表示路径的，例如，资源或文件系统路径。

处理资源与使用 java.io.File 处理文件非常相似。API 是基于句柄的。当用户使用 API 时，例如，getProject 或 getFolder，将对资源返回句柄。在用户试图对句柄执行操作之前，不能保证或要求资源本身是存在的。如果用户期望资源存在，则可以使用 exists()方法来确保资源存在。

要从插件中浏览工作区，必须首先获得 IWorkspaceRoot，它表示工作区中的资源层次结构的顶部：

```
IWorkspaceRoot myWorkspaceRoot = ResourcesPlugin.getWorkspace().getRoot();
```

一旦用户具有了工作区根目录，就可以访问工作区中的项目：

```
IProject myWebProject = myWorkspaceRoot.getProject ("MyWeb") ;
// 如果项目存在并且没有打开，那么打开它。
if (myWebProject.exists() && !myWebProject.isOpen())
  myWebProject.open (null) ;
```

必须打开项目才能进行处理。打开项目时，将从磁盘中阅读项目的结构，并创建项目的资源树的内存对象表示法。打开项目是一种显式操作，原因是每个打开的项目都会消耗内存来在内部表示资源树，而打开的项目会参与各种资源的有效期事件（例如，构建），这可能是一个漫长的过程。一般来说，不能访问关闭的项目，即使资源仍然存在于文件系统中，关闭的项目也将显示为空的。

用户将注意到，当处理资源时，其中许多资源示例都传送 null 参数。许多资源操作都可能会花很长时间，足以使用进度报告和用户取消。如果代码具有用户界面，则用户通常将传送 IProgressMonitor，它允许资源插件在处理资源时报告进度，并且在必要时允许用户取消操作。现在，只传送 null，表示没有进度监视器。

一旦有打开的项目，用户就可以访问它的文件夹和文件，以及创建附加的文件夹和文件。在以下示例中，根据位于工作区外部的文件的内容来创建文件资源。

```
IFolder imagesFolder = myWebProject.getFolder ("images") ;
if (imagesFolder.exists()) {
   // 创建一个新文件
   IFile newLogo = imagesFolder.getFile ("newLogo.gif") ;
   FileInputStream fileStream = new FileInputStream (
      "c:/MyOtherData/newLogo.gif") ;
   newLogo.create (fileStream, false, null) ;
}
```

在以上示例中，第一行获得图像文件夹的句柄。必须核实文件确实存在之后，才能对它执行有意义的操作。同样地，当用户获得文件 newLogo 时，在最后一行创建文件之

前，句柄不能代表实际的文件。在此示例中，可通过利用 logo.gif 的内容来填充文件来创建文件。

以下代码片段类似于前一代码片段，只不过它是根据原始文件复制 newLogo 文件，而不是根据它的内容来创建新文件。

```
IFile logo = imagesFolder.getFile("logo.gif");
if (logo.exists()) {
    IPath newLogoPath = new Path("newLogo.gif");
    logo.copy(newLogoPath, false, null);
    IFile newLogo = imagesFolder.getFile("newLogo.gif");
    ...
}
```

最后，我们将创建另一个图像文件夹，并将新创建的文件移到该文件夹中。作为移动文件的附加效果，我们也对其进行了重命名。

```
...
IFolder newImagesFolder = myWebProject.getFolder("newimages");
newImagesFolder.create(false, true, null);
IPath renamedPath = newImagesFolder.getFullPath().append("renamedLogo.gif");
newLogo.move(renamedPath, false, null);
IFile renamedLogo = newImagesFolder.getFile("renamedLogo.gif");
```

3. 将资源映射至磁盘位置

在本地磁盘的资源树中，我们所有的项目内容目录的默认位置都在平台根目录下面的 workspace 目录中。这是项目的缺省配置。然而，项目的内容目录可以映射到文件系统中的任何目录，甚至可能在另一磁盘驱动器上。映射项目的位置的能力允许用户将项目的内容存储在对项目和项目小组有意义的位置。项目的内容目录可视做“对外开放的”。这意味着用户可以通过使用工作台和插件，或者直接使用基于文件系统的工具和编辑器来创建、修改和删除资源。

资源路径名不是完整的文件系统路径。资源路径始终以项目的位置（通常为 workspace 目录）为基础。要获得指向资源的完整文件系统路径，必须使用 IResource.getLocation 来查询它的位置。但是，用户不能使用 IProjectDescription.setLocation 来更改其位置，因为该方法只是对数据结构的一个简单调节。相反，给定文件系统路径，若用户要获得对应的资源对象，用户可以使用 IWorkspaceRoot.getFileForLocation 或 IWorkspaceRoot.getContainerForLocation。

4. 资源 API 和文件系统

当使用资源 API 来修改工作区的资源树时，除了更新资源对象之外，还会更改文件系统中的文件。在平台的资源 API 之外（例如在文件系统中）发生的资源更改又如何呢？

在资源插件检测到资源的外部更改之前，这些更改不会反映在工作区和资源对象中。客户机可以使用资源 API 来协调工作区和资源对象与本地文件系统，不需要用户干预。用户始终可以在工作台的资源导航器视图中显式地强制执行刷新。

资源 API 中的许多方法都包含一个强制参数，该参数指定应该如何处理与文件系统不同步的资源。每种方法的“API 参考”都提供了有关此参数的特定信息。API 中的其

他方法允许使用程序控制文件系统刷新。

2.3.5.2 资源属性

资源可以具有保存由工具定义的状态信息的属性。资源属性是由各种插件来声明、访问和维护的，并且不是由平台来解释的。当从工作区中删除资源时，还会删除它的属性。有两种资源属性：

1. 会话属性

允许插件高速缓存键值对中的信息。这些值是随机对象。这些属性是在内存中维护的，当从工作区中删除资源或者关闭项目或工作区时，这些属性就会丢失。

2. 持久属性

用来将特定于资源的信息存储在磁盘上。持久属性的值是随机字符串。插件决定如何解释字符串。字符串应该比较短（少于 2 KB）。持久属性与平台元数据一起存储在磁盘上，并在关闭平台和重新启动平台期间进行维护。如果用户具有较大的持久属性，则应该以它们本身的条件而不使用持久属性 API 作为资源显示。

2.4 定制工作台

Eclipse 可以定制工作台的外观和行为等许多方面来满足用户的个人需要。定制工作台操作主要包括以下内容：

- 重新安排主工具栏各项的位置；
- 定制键绑定；
- 更改使用的字体和颜色；
- 更改选项卡的放置；
- 控制单击和双击行为；
- 定制透视图。

2.4.1 定制工具栏

通过定制可以重新安排主工具栏中各项的位置，工具栏各项之间由一条细竖线分隔。定制工具栏的步骤如下：

（1）首先确保工具栏是处于解锁状态的。如果在细的垂直工具栏分隔器旁边具有粗的垂直栏，则工具栏是解锁的。图 2-55 分别显示了工具栏的解锁状态和未解锁状态。



图 2-55 工具栏的解锁和未解锁状态

（2）如果工具栏是被锁定的，则通过选择“窗口”→“锁定工具栏”菜单项来解锁工具栏，如图 2-56 所示。



图 2-56　锁定工具栏

（3）把鼠标放到需要重新安排的工具栏部分左边的粗垂直线上移动。鼠标光标将更改其形状来指示你可以单击以移动工具栏部分。单击并按住鼠标左键以抓住工具栏部分。将该部分左右或上下移动，移到要放置在新位置后释放鼠标键即可。

（4）为了防止对工具栏的意外更改，在定制完工具栏后，通过选择“窗口”→“锁定工具栏”菜单项再次锁定工具栏。

2.4.2　定制键绑定

在 Eclipse 工作台中有广泛地定制键盘的功能，通过指定击键和键序列来调用特定命令。在介绍定制键盘之前，先将一些术语作一解释。

1. 击键、键序列和键绑定

（1）击键——击键是指按键盘上的某一键，同时可选择按住以下一个或多个组合键：Ctrl、Alt、Shift。例如，按住 Ctrl 键而后按下 A 键会组成击键 Ctrl+A。按下这些组合键本身并不构成击键。

（2）键序列——键序列是一个或多个击键。传统上，Emacs（扩展标准）将两个或三个击键键序列指定给特定的命令。例如，在 Emacs 中指定给 Close All 的正常键序列为 Ctrl+X Ctrl+C。要输入此键序列，可按击键 Ctrl+X，随后按击键 Ctrl+C。Eclipse 支持任意长度的键序列。

（3）键绑定——命令与应调用该命令的键组合之间的关联称为键绑定，是将键序列指定给命令。

2. 键配置

键配置是一组键绑定，用来表示键绑定的通用样式或主题。Eclipse 包括两种键配置：

标准配置和 Emacs（扩展标准）配置。

（1）标准配置——包含一组通用键绑定。在许多情况下，可被用户认为是传统的键序列。例如，Ctrl+A 被指定给全选（Select All），而 Ctrl+S 被指定给保存（Save）等。

（2）Emacs 配置——包含 Emacs 的用户熟悉的一组键绑定。例如，Ctrl+X H 被指定给 Select All，而 Ctrl+X S 被指定给 Save 等。了解 Emacs 配置为什么是“扩展标准”很重要。Emacs 配置不是一组完整的键绑定，这一点不像标准配置。但它尽可能地借用标准配置的内容，只是在定义显式可选择的 Emacs 样式键绑定方面不同于标准配置。通常，只有熟知的命令（例如，Select All 和 Save，等等）才具有与它们相关联的特定 Emacs 键序列。

用户通过在“键首选项页面”上更改“活动的配置”设置，来决定哪个配置最好用。如果用户选择标准配置，则忽略所有 Emacs 键绑定。如果用户选择 Emacs 配置，则显式 Emacs 样式键序列指定优先于标准配置中的任何有冲突的指定。

3．作用域

作用域用来为键绑定定义适当的上下文，作用域不同，键绑定会有所不同。工作台定义如下三种键绑定作用域：

- 全局作用域表示键绑定在整个工作台用户界面中都适用；
- 文本编辑器作用域表示仅当文本编辑器活动时，键绑定才适用；
- Java 编辑器作用域表示仅当 Java 编辑器活动时，键绑定才适用。

活动的部件有时可能是一个 Java 文件编辑器。作为特定示例，通常将 Ctrl+B 指定给诸如 Java 文件编辑之类的上下文中的 Build，而将 Ctrl+B 指定给诸如 html 文件编辑之类的上下文中的 Make Text Bold。此上下文称为“作用域”，它由活动部件确定。如果活动部件不选择特定作用域，则工作台会将活动作用域设置为全局。

与配置很相似，作用域可以扩展其它作用域。例如，Java 编辑器作用域从文本编辑器作用域借用键绑定，而后者又从全局作用域借用键绑定。

4．平台和语言环境

键绑定也会随平台和语言环境的不同而有所改变。在 Macintosh 平台上，Command+S（而不是通常的 Ctrl+S）被指定给 Save。在“中文”语言环境（zh）中，Alt+/（而不是通常的“Ctrl+空格”）被绑定给内容援助（Content Assist）。当前平台和语言环境是在 Eclipse 启动时确定的，在 Eclipse 实例的进行过程中不会有改变。

5．定制键绑定

在定制键绑定时要记住很多事项，如上述提到的多击键、键序列、键配置和作用域等事项。为使工作更易于进行，所有键定制都是在“键”首选项页面上完成的。选择“窗口”→“首选项”→“工作台”→“键”以获取“键”的首选项页面，如图 2-57 所示。在该页面中，可以为 Eclipse 中的许多命令指定键序列。

下面给一个示例，了解一下键的一些具体设置。例中选择“Emacs（扩展标准）”作为“活动配置”，并且已从命令（M）的列表中选择“文件”→“关闭”。有关此命令的信息以及它的当前键绑定显示在右边，如图 2-58 所示。



图 2-57　定制键的首选项窗口



图 2-58　绑定键

“关闭”具有两个指定给它的键序列：Ctrl+F4（在标准配置中）和 Ctrl+X K（在 Emacs 配置中）。两个都是在全局作用域中指定的。这样，如果用户将活动配置设置为“标准”，则会将 Ctrl+F4（而非 Ctrl+X K）指定给“关闭”。反过来，如果用户将活动配置设置为“Emacs”，则会将 Ctrl+X K 指定给“关闭”。另外，由于 Emacs 配置也从标准配置

借用键绑定，所以也会将 Ctrl+F4 指定给“关闭”（假定 Ctrl+F4 尚未被指定给 Emacs 配置中的另一个命令）。

在指定给“关闭”的键序列的列表下面，有一个地方用来添加或除去键绑定。默认情况下，它选择的作用域为全局，而选择的配置则为用户活动配置。在“键控顺序（K）”文本框内输入键序列 Ctrl+W，这时“添加”按钮就变为启用的。另外，已对其指定键序列 Ctrl+W 的所有命令的列表都显示在“添加”按钮下面。此时可以看到 Ctrl+W 当前被指定给全局作用域和 Emacs 配置中的命令 Cut。单击“添加”按钮将 Ctrl+W 指定给“关闭”。

现在可看到 Ctrl+W 被指定给“关闭”的键序列的列表，如图 2-59 所示。



图 2-59 “关闭”快捷键的设置

6. 键绑定的动态性质

键绑定是由插件提供的，在 Eclipse 中，可以添加或除去插件。这会导致添加或除去由这些插件所声明的键绑定。Eclipse 以某种方法存储定制键绑定以自动补偿这一点。考虑上面的示例，Ctrl+W 已被指定给 Emacs 配置中的“关闭”。考虑用户安装新的插件，该插件将 Ctrl+W 指定给其它特定命令。Eclipse 将保留用户对“关闭”的指定，但是显示带有“更改”图标（）而不是“加号”图标（）的键绑定。

7. 冲突解决

只有有限数量的简单的常用击键可用于指定给众多命令。现在看到可以通过键配置、作用域、平台和语言环境，来将键序列指定到相互不冲突的域中。如果作用域不存在，可考虑前面提到的 Ctrl+B 的情况。一个插件会将 Ctrl+B 指定给 Build，而另一个插件则会将 Ctrl+B 指定给 Make Bold Text。Eclipse 将如何正确解决此冲突呢？

虽然通过应用上面的机制大大减少了冲突，但是冲突仍然会发生。两个插件（相互无关）可将同一键序列指定给具有相同作用域、配置、平台和语言环境的不同命令。考虑插件是否已在全局作用域和标准配置中将 Ctrl+F4 指定给它的其中一条命令。这与在同一作用域和配置中将 Ctrl+F4 指定给“关闭”命令的 Eclipse 直接冲突。

同时调用这两条命令是不正确的，只选择这两条命令中的其中一条来接收击键也是不正确的。唯一正确的做法是忽略两个键绑定，使 Ctrl+F4 在此作用域和配置中确实不起作用。这些冲突类型可以由用户显式地将键序列指定给其中一条命令来解决，如图 2-60 所示。

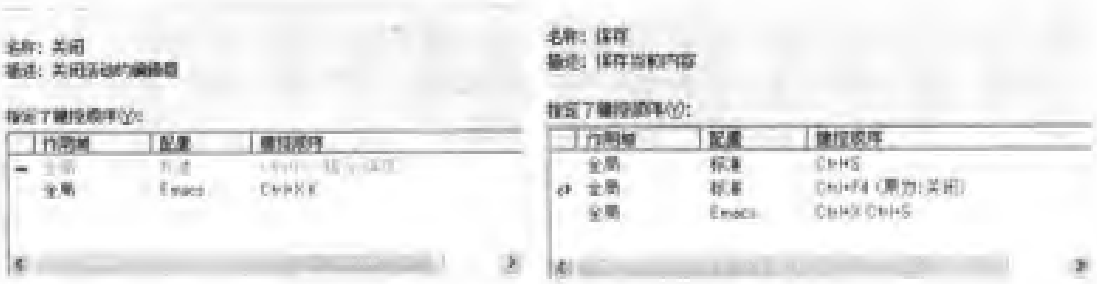


图 2-60　键绑定的冲突解决

另一种冲突类型可由多击键键序列导致。例如，在 Emacs 配置中，有许多多击键键序列以击键 Ctrl+X 开头。Ctrl+H K 被指定给“关闭”。Ctrl+X H 被指定给“全选”。如前面所提到的，Emacs 配置从标准配置借用键绑定。在标准配置中，Ctrl+X 被指定给“剪切”。虽然 Emacs 配置不显式地重定义 Ctrl+，但是按 Ctrl+X（作为该配置的许多键绑定的一部分）是必需的。在 Emacs 配置中，Ctrl+X 只是许多指定键序列中的一个，而不仅仅是调用“剪切”操作。

对于此冲突类型，规则是将忽略指定给“剪切”的 Ctrl+X 键序列。否则，就不可能完成 Emacs 配置中的许多键绑定。

2.4.3　更改字体和颜色

缺省情况下，工作台使用操作系统提供的字体和颜色。Eclipse 允许用户更改字体和颜色。

1. 字体

工作台允许直接配置以下三种字体。

- 标题字体：用在 PDE 编辑器、欢迎页面和许多向导的标题区域中。例如，“新建项目”向导将此字体用于顶部标题。
- 页眉字体：用做小节标题。例如，Eclipse 平台的“欢迎页面”将此字体用于顶部标题。
- 文本字体：用在文本编辑器中。

更改这些字体的具体步骤如下：

（1）在主菜单栏上，单击“窗口”→“首选项”→“工作台”→“字体”，如图 2-61 所示。

（2）选择想要更改的字体，单击“更改”。

（3）使用打开的对话框来选择字体。

（4）单击“应用（A）”。

注意：还可以单击“使用系统字体”来将字体设置为操作系统选择的合理值。例如，在 Windows 上，这将使用在“显示属性”控制面板中选择的字体。

使用其他字体的插件还可以提供首选项条目以便可以定制它们。例如，“Java 开发工具”提供了用于控制 Java 编辑器使用的字体的设置。在首选项画面中选择“Java”→“编辑器”→“外观”→“文本字体”来进行设置。

除了以上各项之外，某些文本始终是用系统字体显示的。例如，导航器树始终就是这样。要更改在这些区域中使用的字体，可以使用操作系统提供的配置工具（例如，Windows 上的“显示属性”控制面板或 Motif 中的.Xdefaults 文件）。



图 2-61　字体设置首选项窗口

2. 颜色

要设置工作台用来显示错误文本和超级链接文本的颜色，其步骤如下：

（1）在主菜单栏上，单击“窗口”→“首选项”→“工作台”→“外观”，如图 2-62 所示。

（2）选择想要更改颜色的项（例如，错误文本、超级链接文本、活动的超级链接文本）。

（3）使用打开的对话框来选择颜色。

（4）单击“应用（A）”。

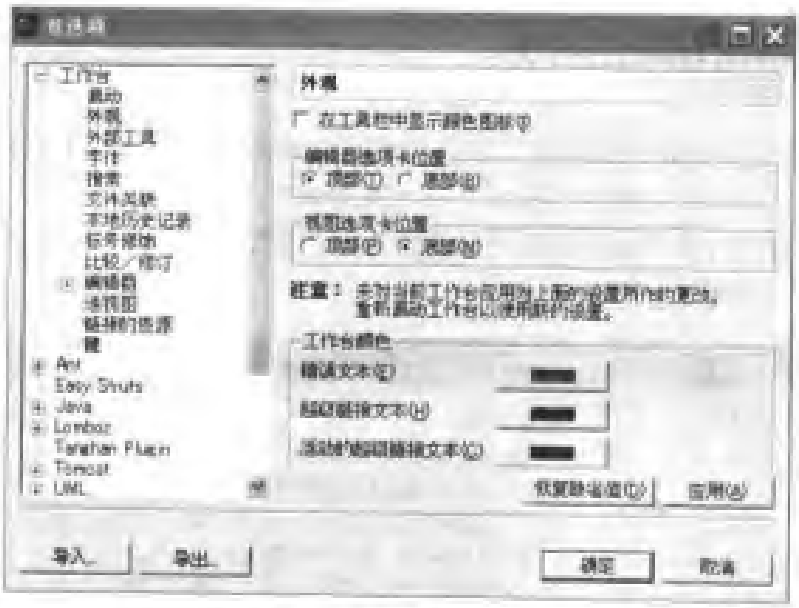


图 2-62　颜色设置首选项窗口

使用其他颜色的插件还可以提供首选项条目以便可以定制它们。例如，搜索支持提供了用于控制用来显示非精确匹配的颜色的设置。在首选项画面中选择“工作台”→“搜索”→“不精确匹配的前景色”中进行设置。

通常，工作台使用由操作系统选择的颜色。要更改这些颜色，可以使用系统提供的配置工具（例如，Windows 上的“显示属性”控制面板或 Motif 中的 Xdefaults 文件）。

2.4.4　更改选项卡的位置

可以更改选项卡的位置。叠放的视图或编辑器的选项卡可以出现在包含它们的区域的顶部或底部。操作步骤如下：

（1）在主菜单栏上，单击“窗口”→“首选项”→“工作台”→“外观”；

（2）使用“编辑器选项卡位置”组或“视图选项卡位置”组中的单选按钮来控制这些选项卡位于顶部还是底部，如图 6-63 所示；

（3）单击“应用（A）”；

（4）关闭并重新打开工作台可以查看更改情况。

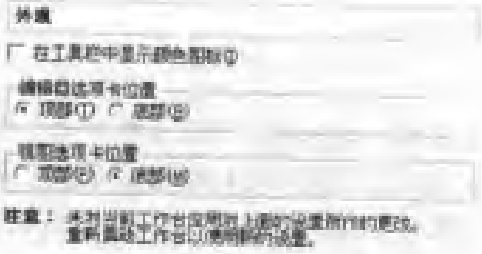


图 6-63　更改选项卡的放置

2.4.5 控制单击和双击行为

Eclipse 还可以通过定制来控制工作台如何响应单击和双击，步骤如下：

（1）在主菜单栏上，单击“窗口”→“首选项”→“工作台”；

（2）从“打开方式”选择想要更改的行为，如图 2-64 所示；

（3）单击“应用（A）”。



图 2-64 控制单击和双击行为

这些选择的效果随视图的不同而不同。例如，在导航器中，选择“双击”打开方式时，将导致单击资源仅是选中它，而只有双击资源时，才能在编辑器中打开它；选择“单击”打开方式将导致单击资源就能选中它并立即在编辑器中打开它。

选择“单击”的单选按钮后，下面的复选框进一步优化单击行为。选择“根据悬浮提示选择（H）”将导致将鼠标悬浮在资源上就会选择它。选择“使用箭头键时打开（K）”将导致使用箭头键导航至资源时就打开该资源。

2.4.6 定制透视图

每个透视图都包括预定义的一组操作，可从菜单栏和工作台工具栏访问这些操作。相关操作分成一些操作集。此命令让用户可以通过显示或隐藏各个操作集来定制当前透视图。图 2-65 所示的前三项（“文件 > 新建”、“窗口 > 打开透视图”、“窗口 > 显示视图”）是来控制哪些操作作为顶级项出现在它们各自的菜单中。最后一项（“其它”）是来控制哪些操作集在透视图中是可视的。

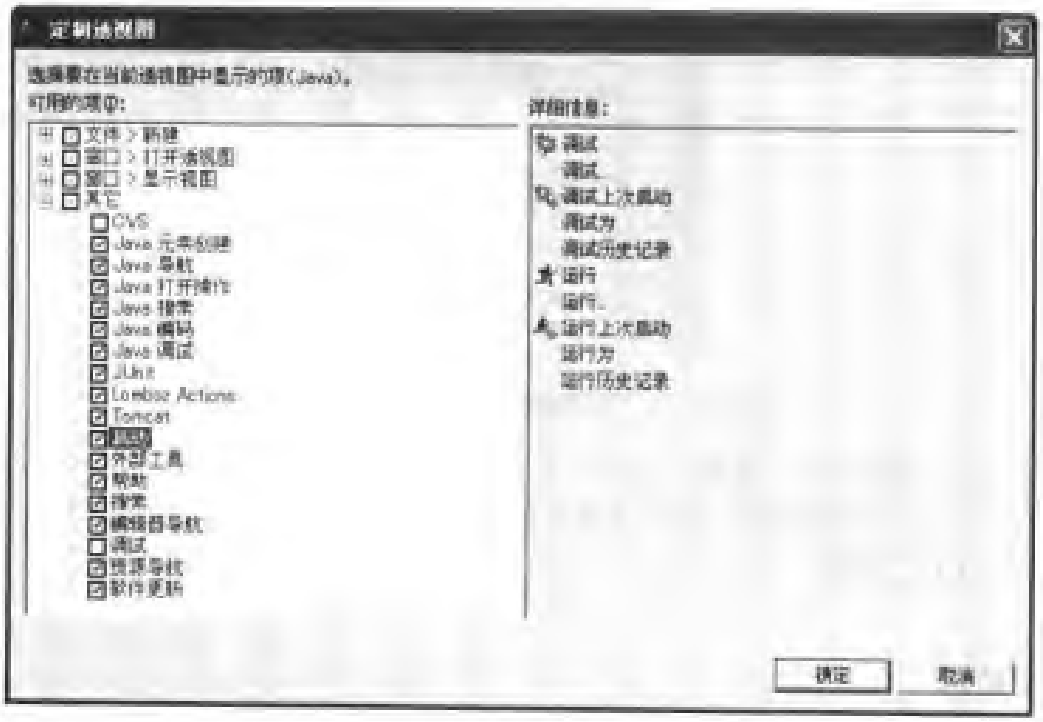


图 2-65　定制透视图窗口

1. 显示在工具栏上的操作集

在快捷方式栏中单击“资源”透视图。选择“窗口”→“定制透视图”→“其它”。选择“启动”多选框，并单击“确定”。

这时会发现在工具栏上多出了两个包括用于调试和运行的按钮，如图 2-66 所示。



图 2-66　工具栏上的调试和运行按钮

2. 设置顶级项视图

选择“窗口”→“定制透视图”→“窗口 > 显示视图”。选择“包资源管理器”多选框，并单击“确定”。你会发现在工作台上会出现“包资源管理器”的透视图，如图 2-67 所示。

3. 更改打开透视图的位置

用户可以更改如何在工作台中打开透视图的缺省行为。可以选择在同一窗口中或新开窗口中打开透视图，其步骤如下。

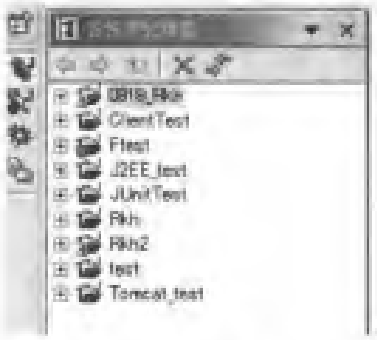


图 2-67　设“包资源管理器”为顶级项视图

（1）从主菜单栏中，选择“窗口”→“首选项”→“工作台”→“透视图”；
（2）从“打开新透视图”组中选择“在同一窗口中”或“在新窗口中”；
（3）单击“确定”。

本章小结

本章主要介绍了 Eclipse 平台的插件体系结构、工作台用户界面基本操作以及资源管理等知识。通过本章的学习，读者不仅可以全面地了解 Eclipse 这一集成开发环境，包括工作台 UI、工作区、运行时核心等平台重要组成部分的技术概况，还能熟悉并掌握 Eclipse 工作台用户界面的基本使用。在下面的章节中，将继续详细介绍 Eclipse 主要插件的使用及其开发知识。

第二篇

Eclipse 编程工具

PART 2

第3章 利用 JDT 编写 Java 程序

第4章 利用 CDT 编写 C/C++程序

第3章　利用JDT编写Java程序

“Java开发工具”（JDT）是随Eclipse平台一起交付的全功能的Java IDE。JDT是对工作台的一组扩展，它允许用户编辑、编译和运行Java程序。最简单的方法是将JDT看作是一组插件，这些插件将特定于Java的行为添加到一般平台资源模型中，并向工作台添加特定于Java的视图、编辑器和操作。对于初次接触Eclipse的读者来说，可以把JDT看作和JBuild有着很多类似的Java程序开发工具。但是随着对它的深入了解，读者会发现它是功能强大、最有发展前景的编程工具，这完全得益于Eclipse插件式的体系结构。

本章将介绍JDT的基本组成和使用方法。通过本章的学习，读者就能够灵活自如地应用Eclipse平台进行Java程序的编辑、调试及运行。本章主要介绍的内容有：

- JDT的相关操作
- JDT的相关视图
- Java编辑器
- JDT调试器
- Java搜索
- Java构建器

3.1　JDT的基础操作

JDT是对工作台的一组扩展，它们允许编辑、编译和运行Java程序。用户可以通过菜单栏、工具栏及视图中的上下文菜单来获得JDT操作。

JDT操作具体包括文件操作、编辑操作、源操作、重构操作、浏览操作、搜索操作、项目操作、运行操作、Java工具栏操作、Java编辑器操作、运行和调试操作。

3.1.1　验证JRE安装和类路径变量

如果读者第一次使用JDT，首先需要验证是否为Java开发正确地设置了Eclipse工作台。如果你不熟悉基本工作台机制，请参阅第2章中的Eclipse工作台的详细介绍。以下是检查工作台是否正确设置的步骤：

（1）选择菜单项的“窗口”→“首选项”来打开工作台的首选项。

（2）选择左边的树窗格中的“Java”→“已安装的JRE”来显示已安装的Java运行时环境首选项页面，如图3-1所示，确认检测到JRE。缺省情况下，用来运行工作台的

JRE 将用来构建和运行 Java 程序。它应与安装的 JRE 列表中的选择标记一起出现。建议你使用 Java SDK 而不是 JRE。SDK 是为开发设计的，它包含 Java 库的源代码，因而简化了调试。可以通过在硬盘驱动器上搜索已安装的 SDK 来添加附加的 SDK。为此，只需单击“搜索（S）...”按钮并指定搜索的根文件夹。



图 3-1　检测已 JRE 安装

（3）选择树窗格中的“工作台”选项，显示工作台首选项页面，如图 3-2 所示，请确认选择了“对资源修改时自动执行构建（B）”选项。

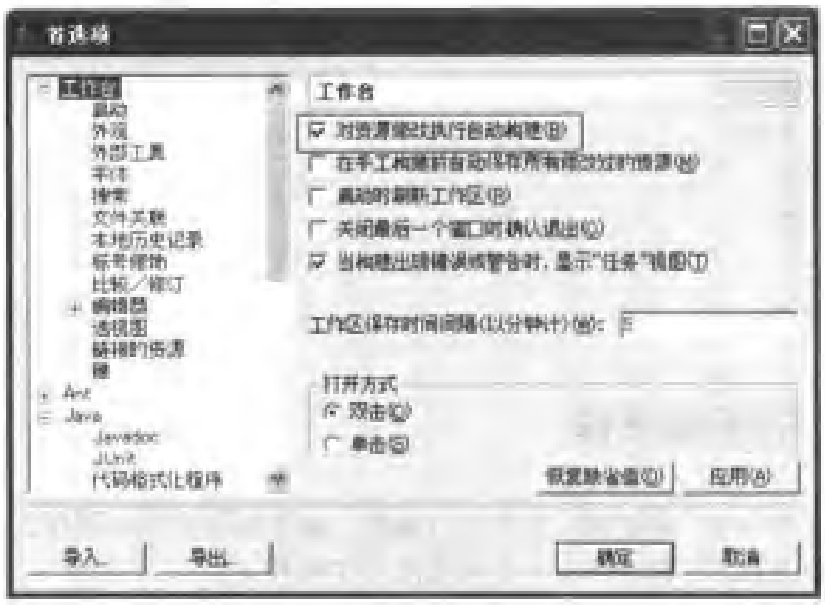


图 3-2　工作台首选项窗口

（4）在树窗格中选择“Java”→“新建项目”，显示“新建项目”首选项页面，如图 3-3 所示，确认将源和输出文件夹设置为“项目（P）”。在这种组织中，所有 Java 包是直接在项目内创建的，缺省情况下就选择这种组织。生成的 CLASS 文件与 Java 源文件存储在一起，对于简单的项目，建议使用此组织。但是对于复杂的项目来说，则应该选择“文件夹（F）”的单选框，系统自动将源文件（即*.java 文件）存放到 src/文件夹下，而将输出文件（即*.class 文件）存放到 bin/文件夹下。

（5）在树窗格中选择“Java”→“编辑器”，显示“Java 编辑器”首选项页面。在编辑器的首选项页面上（如图 3-4 所示），按“注释（S）”选项卡，设置选项完成后，单击“确定”以保存首选项。

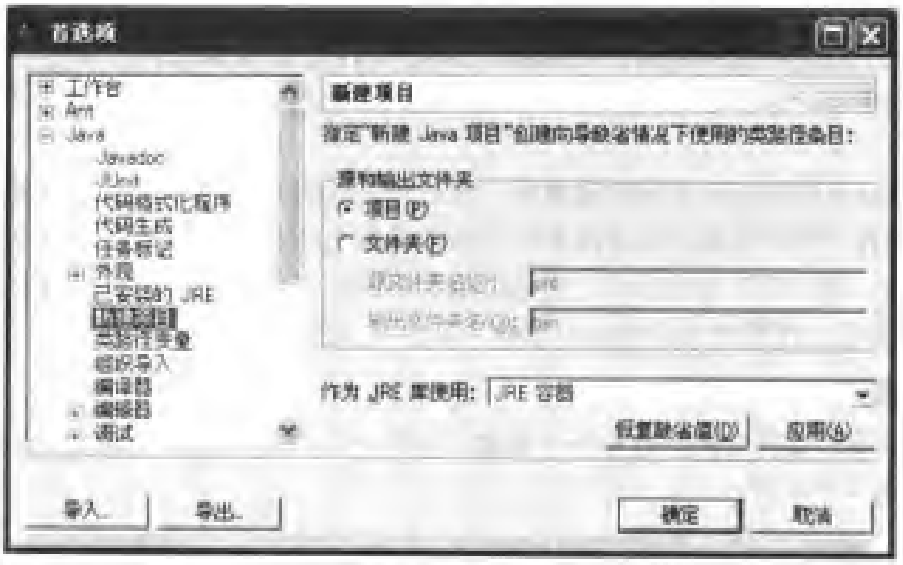


图 3-3　新建项目

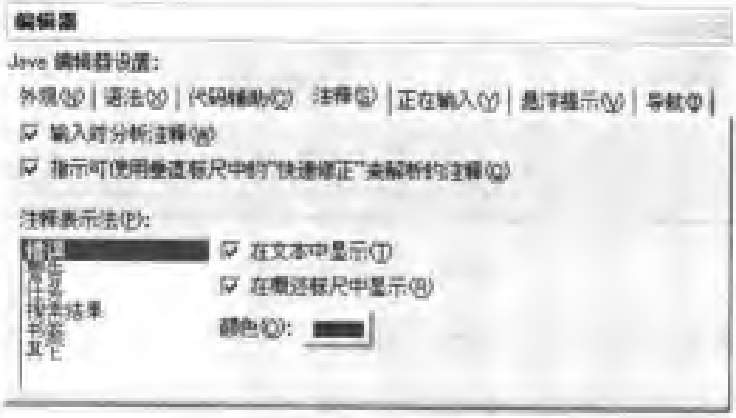


图 3-4　Java 编辑器外观设置

3.1.2　创建 Java 项目

在本节中，通过创建一个新的 Java 项目，介绍 JDT 的基本功能及使用方法。

1. Java 项目的相关概念

Java 项目包含用于构建 Java 程序的源代码和相关文件。它有一个相关联的 Java 构建器，在更改 Java 源文件时，Java 构建器可以对这些 Java 源文件进行增量编译。

Java 项目还维护其内容的模型。此模型包含关于 Java 元素的类型层次结构、引用和声明的信息。当用户更改 Java 源代码时，此信息会不断地自动更新。内部 Java 项目模型的更新与 Java 构建器无关；尤其是在执行代码修改时，即使关闭了自动构建，模型也仍将反映最新的项目内容。

可以用以下两种不同方法来组织 Java 项目。

- 将项目用做源容器。在此项目组织中，所有 Java 包是直接在项目内创建的。缺省情况下就选择这种组织。生成的 CLASS 文件与 Java 源文件存储在一起。对于简单的项目，建议使用此组织。可参考图3-3，在新建项目画面中选择“项目”单选框。
- 在项目内将源文件夹用作源容器。在此项目组织中，包不是直接在项目下而是在源文件夹中创建的。创建源文件夹来作为项目的子目录，并在这些源文件夹中创建包。对于较复杂的项目，建议使用此组织。它允许你将包划分成多个组。如图 3-3 所示，在新建项目画面中选择“文件夹（F）”单选框。

2. 新建 Java 项目步骤

（1）选择“文件（F）”菜单→“新建（N）”→“项目（R）”，选择“Java”→“Java 项目”，如图 3-5 所示。



图 3-5 新建 Java 项目

或者通过如图 3-6 所显示的工具栏中的按钮，来更快捷地创建 Java 项目。

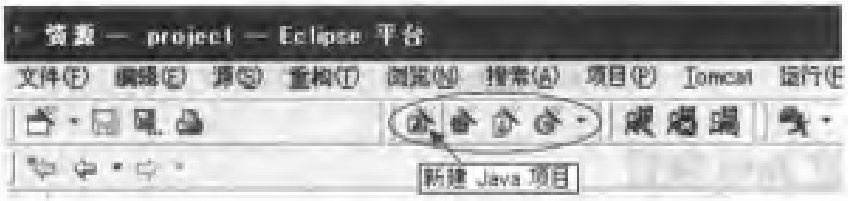


图 3-6 新建 Java 项目工具栏

（2）单击“下一步”，出现如图 3-7 所示的画面。在此画面的“项目名”文本框中输入项目名“JDT-ch3”。项目内容建议选择“使用缺省值（D）”，如果你的 Eclipse 是建在 D:\eclipse 目录下，则该项目将自动保存在 D:\eclipse\workspace\ JDT-ch3 目录下。

（3）单击“完成”后，就会在 Eclipse 上的“Java”透视图的“包资源管理器”视图中看到新创建了一个名为“JDT-ch3”的项目，如图 3-8 所示。在此视图中，用户还可以浏览该项目中的 Java 元素及为该项目所配置的 JRE 文件。Java 元素包括 Java 项目、包、编译单元、类文件、类型、方法和字段。



图 3-7　创建新的 Java 项目



图 3-8　“包资源管理器”视图

3.1.3　新建 Java 类

下面使用代码生成操作来创建新的 Java 类。

（1）先选中“JDT-ch3”项目，选择菜单中的“文件（F）”→“新建（N）”→“类”，或者也可以直接在工具栏里单击“创建 Java 类”的按钮（带有一个 C 标记），如图 3-9 所示。



图 3-9　新建 Java 类的工具栏

（2）在弹出的新建 Java 类窗口里输入类的名称“HelloWorld”。在这里包名为“com.dkd.jdt”，然后，通过在“你想要创建哪些方法存根？”的复选框里选中“public static void main(String[] args)”的选项来自动创建一个 main 方法，如图 3-10 所示。

（3）单击“完成”后，一个 Java 编辑窗口就会被打开，如图 3-11 所示。

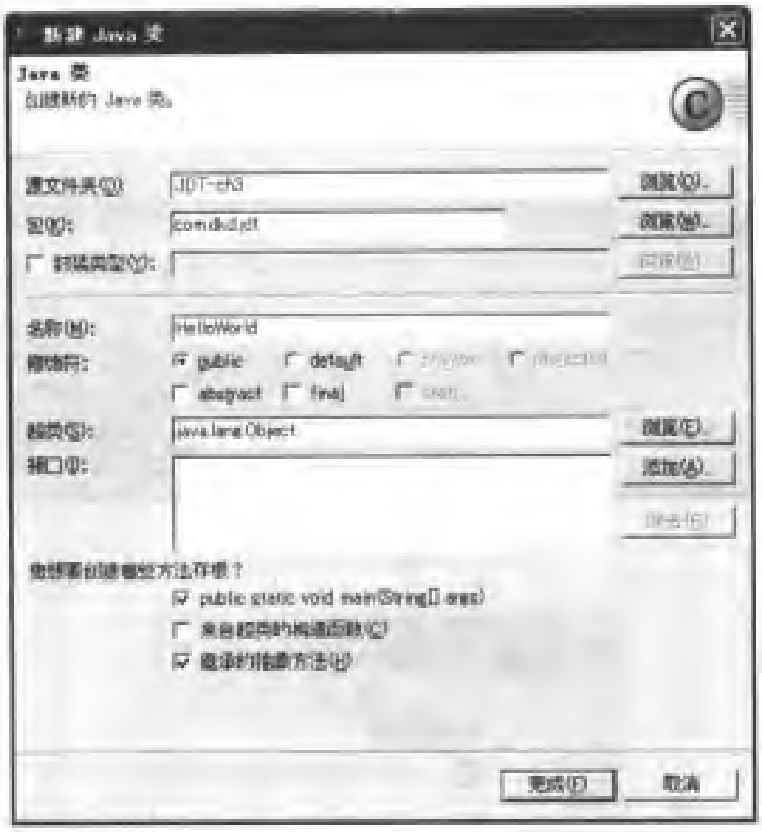


图 3-10　新建 Java 类窗口



图 3-11　Java 编辑窗口

3.2　Java 编辑器

Java 编辑器提供了用于编辑 Java 代码的特殊功能部件。Eclipse 可以对二进制的 CLASS 文件打开 Java 编辑器。如果 JAR 文件包含具有连接源的 CLASS 文件，则编辑器

会显示相应的源文件。

Java 编辑器包含下列功能部件：

- 语法突出显示
- 内容 / 代码辅助
- 代码格式化
- 导入辅助
- 快速修正
- 集成的调试功能部件

3.2.1　打开 Java 编辑器

1. Java 编辑器打开方式

选择文件“HelloWorld.java”，可以通过双击文件在 Java 编辑器中打开，也可以通过单击鼠标右键来选择打开文件类型，如图 3-12 所示。

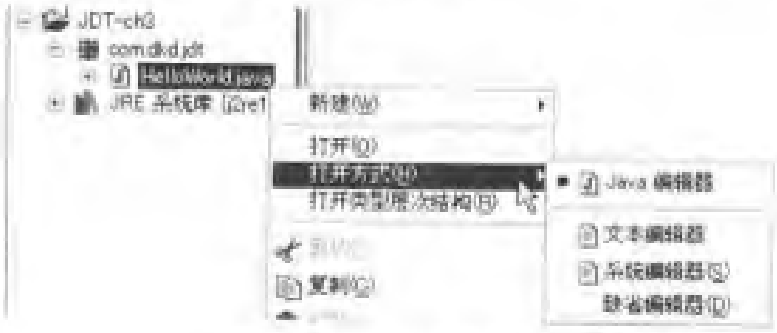


图 3-12　打开 Java 编辑器

2. 语法的高亮显示

在图 3-11 中，可以看到 Java 编辑器的语法突出显示情况。Java 源程序中的不同种类的元素使用独特的颜色显示。以不同方式显示的 Java 元素的示例包括：

- 常规注释
- Javadoc 注释
- 关键字
- 字符串

3. 添加新方法

通过在 Java 编辑器中的“HelloWorld.java”文件中输入下列内容来添加方法：public static String getDayOfWeek(String strYMD)，一旦用户在编辑器区域中输入了方法名，新方法的名称就会出现在“大纲”视图的底部，如图 3-13 所示。

4. 显示错误注释

错误注释（红色的方块）显示在位于编辑器右边的概述标尺中。此错误注释指示编

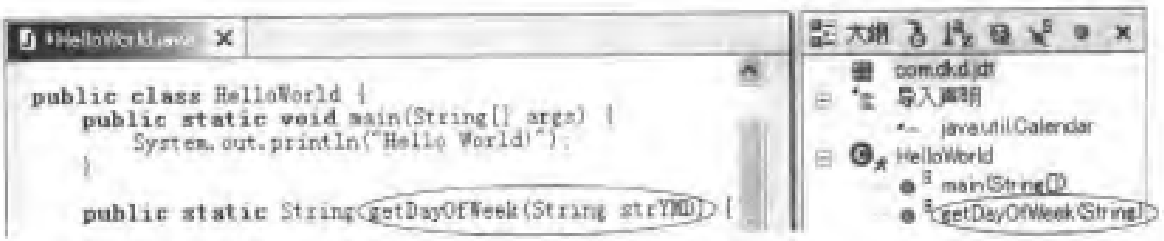


图 3-13 大纲视图

译单元当前不正确。如果将鼠标指针悬浮在红色方块上，则将出现错误提示信息。注意，编辑器标尺中的错误注释会随着用户的输入而更新。接下来在新添加的方法中编写如下所示的代码段。

例 3-1：一个根据年月日来判断是星期几的小方法。

```
public static String getDayOfWeek(String strYMD) {
    Calendar calendar = Calendar.getInstance();
    calendar.set(
            Integer.parseInt(strYMD.substring(0, 4)),
            Integer.parseInt(strYMD.substring(4, 6)) - 1,
            Integer.parseInt(strYMD.substring(6, 8)));
    int intWeek = calendar.get(Calendar.DAY_OF_WEEK);
    String strReturn = "";
    switch (intWeek) {
            case Calendar.SUNDAY :
                    strReturn = "Sunday";
                    break;
            case Calendar.MONDAY :
                    strReturn = "Monday";
                    break;
            case Calendar.TUESDAY :
                    strReturn = "Tuesday";
                    break;
            case Calendar.WEDNESDAY :
                    strReturn = "Wednesday";
                    break;
            case Calendar.THURSDAY :
                    strReturn = "Thursday";
                    break;
            case Calendar.FRIDAY :
                    strReturn = "Friday";
                    break;
            case Calendar.SATURDAY :
                    strReturn = "Saturday";
                    break;
    }
    return strReturn;
}
```

3.2.2 使用内容辅助功能编程

下面讲解使用内容辅助功能来完成新方法的编写工作。

1. 代码辅助功能

在 Java 编辑器中打开“HelloWorld.java”文件，在 main()方法中添加“System.out.println("Hello World!");”语句。在输入“System.”的时候，Eclipse 可以自动弹出建议列表，如图 3-14 所示。在滚动该列表的过程中，用户可以查看可用选项。

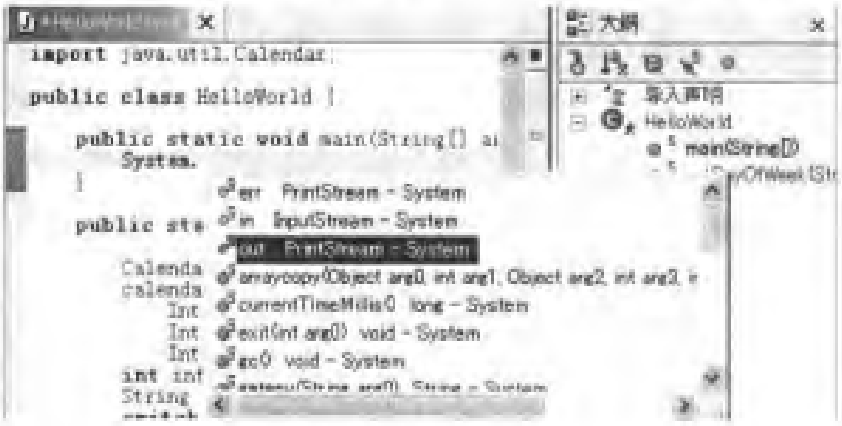


图 3-14　内容辅助功能

另外一种内容辅助功能是编程人员只需输入“sy”，这时通过菜单“编辑（E）”→“内容辅助（N）”或者按“Ctrl+空格”快捷键的方式，同样会出现一个建议列表，选中你所需的项目按回车键即可，此功能对于提高编程速度和代码准确性有很大的帮助，如图 3-15 所示。

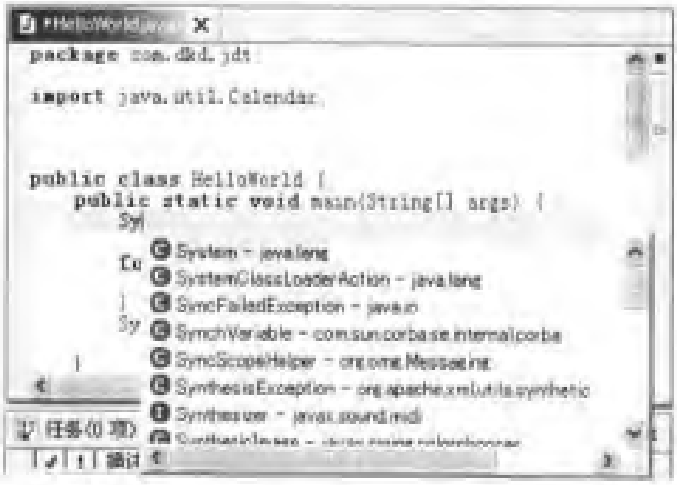


图 3-15　内容辅助功能

2. 生成 getter 和 setter

Java 编辑器允许你为编译单元内的某种类型的字段生成“getter 和 setter”。这里要说明的是不是所有类型的字段都能够生成“getter 和 setter”，如果读者有过开发 Bean 的经验的话，就会充分体验到使用这一功能的便利性。

实现该方法很简单，选择想要为其生成“getter 和 setter”的字段，从“源（S）”弹

出菜单中选择“生成 Getter 和 Setter”，将打开一个对话框，在该对话框中用户可以选择想要创建哪些方法，选择这些方法，并按“确定”完成。

3.2.3 快速修正功能

Java 编辑器提供了对输入时和编译后发现的问题的修正功能。读者如果将例 3-1 中的 getDayOfWeek(String strYMD)方法的源代码完整地输入后，将会看到源代码的垂直标尺上标记着许多问题标识。

为了显示有可用于问题或警告的更正建议，编辑器的注释栏将“问题标识”显示为“灯泡”形式。有以下三种方式可以进行快速修正：

- 用鼠标单击垂直标尺中的灯泡形式的问题标记；
- 通过菜单栏“编辑（E）”→“快速修正（Q）”；
- 按“Ctrl+1”快捷键方式。

采用上述任何一种方式，就会弹出修正问题的建议列表，提示你如何正确地修改这个错误，如图 3-16 所示。系统提供了五种修改错误的方法，第一种方式为最佳修改方法，用户只需按回车键就能够自动修改程序。你会看到系统已经自动在程序的开头部分添加了一行代码“import java.util.Calendar;”，而且所有“问题标识”也会随之消除。

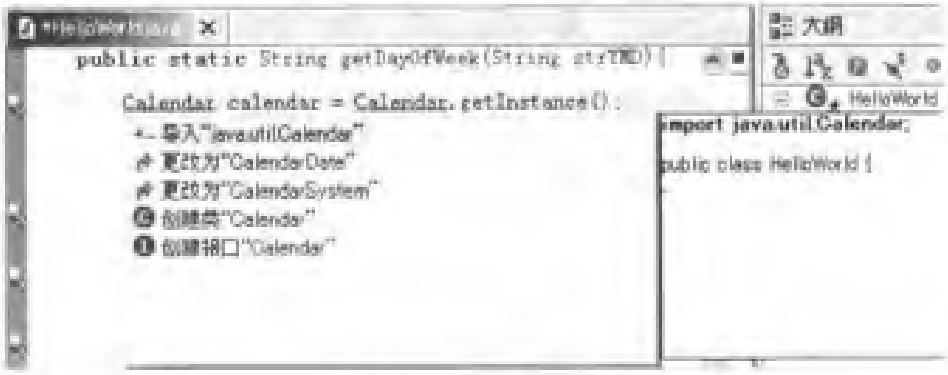


图 3-16 快速修正功能

1. 快速修正

JDT 提供了下列快速修正。

（1）包声明：

- 添加丢失的包声明或更正包声明；
- 将编译单元移至与包声明相对应的包。

（2）导入：

- 除去未使用的、不可解析的或不可视的导入；
- 对于导入中的问题调用“组织导入”。

（3）类型：

- 对于不能解析的类型的引用，创建新的类或接口；

- 更改所访问的但是不可视的类型的可视性；
- 对于不能解析的类型的引用，重命名为相似类型；
- 对于不能解析但却存在于项目中的类型，添加 import 语句；
- 对于歧义类型引用，添加显式的 import 语句（同一类型的两个以上的 import 语句，则按要求导入）；
- 如果类型名与编译单元名不匹配，则重命名类型或重命名编译单元；
- 除去未使用的私有类型。

（4）构造函数：

- 对于不能解析的构造函数的引用，创建新的构造函数（创建 this、super 或 new 类）；
- 对于参数不匹配的构造函数引用，重新排序或除去自变量；
- 将具有构造函数名的方法更改为构造函数（除去返回类型）；
- 更改所访问的但却不可视的构造函数的可视性；
- 除去未使用的私有构造函数；
- 当对父类的构造函数进行调用时，如果该构造函数未定义、不可视或抛出异常时，则在父类中创建该构造函数；
- 如果类型包含未实现的方法，则将类型修饰符更改为“抽象”或添加要实现的方法。

（5）方法：

- 对于不能解析的方法的引用，创建新方法；
- 对于不能解析的方法的引用，重命名为相似方法；
- 对于参数不匹配的方法引用，重新排序或除去自变量；
- 更正所引用的方法的访问（可视性和静态）；
- 除去未使用的私有方法；
- 对于缺少返回类型或返回类型与返回语句不匹配的方法，更正返回类型；
- 如果缺少返回语句，则添加它；
- 对于不具有主体的非抽象方法，更改为“抽象”方法或添加主体；
- 对于非抽象类型中的抽象方法，除去方法的“抽象”修饰符或使类型为“抽象”类型；
- 对于带有主体的抽象 / 本机方法，除去抽象或本机修饰符或除去主体；
- 如果在构造函数内调用方法，则将方法访问更改为“static”（super 和 this）；
- 将方法访问更改为缺省访问以避免仿真方法访问。

（6）字段和变量：

- 更正所引用的字段的访问（可视性和静态）；
- 对于不能解析的变量的引用，创建新字段、参数或局部变量；
- 对于不能解析的引用，重命名为具有相似名称的变量；
- 除去未使用的私有字段；
- 更正静态字段的非静态访问；
- 对于在外部类型中访问的局部变量，添加“final”修饰符；

- 将字段访问更改为缺省访问以避免仿真方法访问；
- 更改局部变量类型以修正类型不匹配；
- 初始化尚未初始化的变量。

（7）异常处理：

- 除去不需要的 catch 块；
- 通过用 try/catch 括起来或周围的 try 块添加 catch 块来处理未捕获的异常；
- 通过将 throw 声明添加至父方法或通过将现有 throw 声明一般化来处理未捕获的异常。

（8）其他：

- 添加强制转型或更改强制转型以修正类型不匹配；
- 对于非 NLS 字符串，打开 NLS 向导或标记为非 NLS。

2. 快速辅助

即使在没有问题或警告时，即，没有显示，“灯泡”图标时，“快速辅助”也可提供有用的建议，因此用户必须了解以下可用的辅助：

- 将 catch 子句替换为父方法中的 throws 声明；
- 将表达式语句转换为对新局部变量或新字段的赋值；
- 除去外围的“if”，“for”，“while”和“do”语句；
- 除去带有单个 catch 或 finally 块的外围 try 块；
- 除去外围的匿名类、块或括号；
- 对处于链接“模板”方式的类型、方法或变量进行本地重命名；
- 用模板（“if”和“while”等）将选择的行括起来。

3.2.4 管理导入语句

1. 添加所需的导入语句

Java 编辑器可以帮助用户添加编译单元内的所选类型所必需的导入语句。选择对 Java 代码中的一种类型的引用，可执行下列任何一种操作方式：

- 按鼠标右键，从编辑器弹出的上下文菜单中选择“源（S）”→“添加导入（D）”。
- 从菜单栏中选择“源（S）”→“添加导入（D）”。
- 按“Ctrl+Shift+M”快捷键。

编辑器可标识该类型或提示你从可能类型的列表中选择期望的类型。按照导入次序首选项指定的那样生成和插入导入语句。

2. 组织现有的导入语句

Java 编辑器可以帮助你改进编译单元内现有的导入语句。在编辑 Java 代码时执行下列其中一项操作：

- 从编辑器的弹出菜单中选择“源（S）”→“组织导入（G）”。
- 从菜单栏中选择“源（S）”→“组织导入（G）”。

- 按“Ctrl+Shift+O”快捷键。

Java 编辑器将按照“组织导入”首选项指定的导入次序生成导入语句的完整列表，并且新的导入语句将代替旧的导入语句。

3. 设置导入语句的次序

对于要求完美的编程人员，可以自己来设置导入语句的次序，方法如下：

（1）从菜单栏中，选择“窗口（W）”→“首选项（P）”。

（2）在左窗格中展开“Java”类别，并选择“组织导入”。

（3）“组织导入”页面定义导入语句的排序次序。在“导入”列表中（如图 3-17 所示），按如下所示项来管理包前缀的列表：

- 若要添加新的前缀，选择“新建（N）”。
- 若要更改现有前缀的名称，选择“编辑（E）”。
- 通过使用“向上（U）”和“向下（W）”按钮，将所选前缀上下移动来重新安排列表的顺序。
- 若要从列表中除去所选前缀，选择“除去（R）”。
- 使用“装入（L）...”按钮将从文件中装入前缀列表。
- 使用“保存（V）...”按钮将前缀列表存储到文件中。

（4）在“.*所需的导入数：”字段中，输入在使用<package prefix>.*之前允许引用同一个包的 import 语句的数目。

（5）完成后单击“确定”，以保存设置。

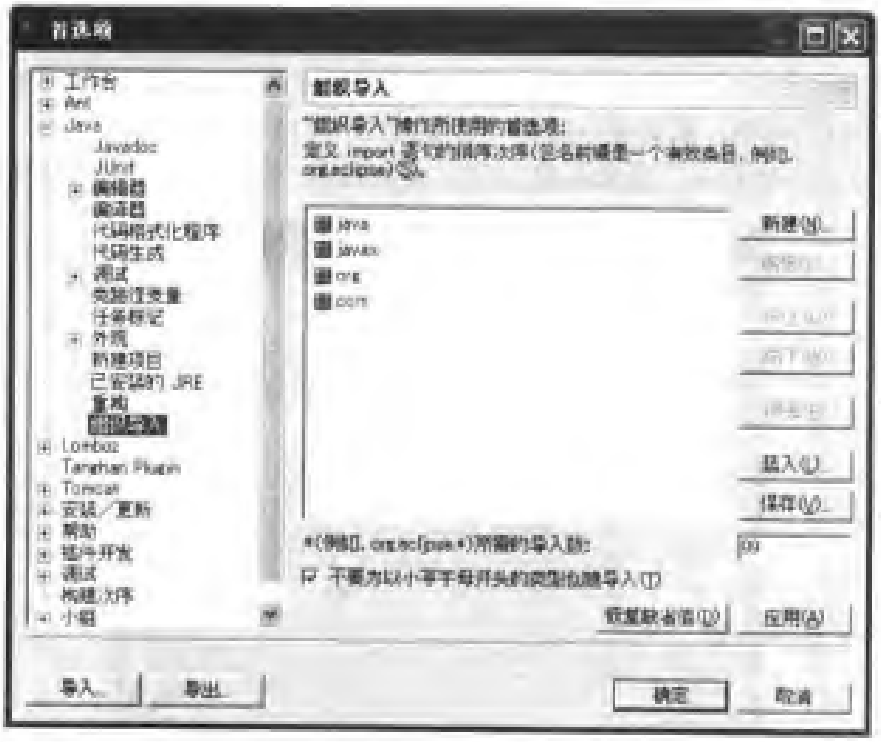


图 3-17　组织导入

3.2.5 创建和查看 Javadoc 文档

1. 创建 Javadoc 文档

（1）指定 Javadoc 命令的位置。JDT 使用 Javadoc 命令（通常在 JRE 分发中使用）根据源文件来生成 Javadoc 文档。要设置 Javadoc 命令的位置，需进行以下操作：

打开菜单项的“窗口”→“首选项”→“Java”→“Javadoc”，在首选项页面中输入 Javadoc 命令的绝对路径（例如，C:\j2sdk1.4.1_03\bin\javadoc.exe），如图 3-18 所示。

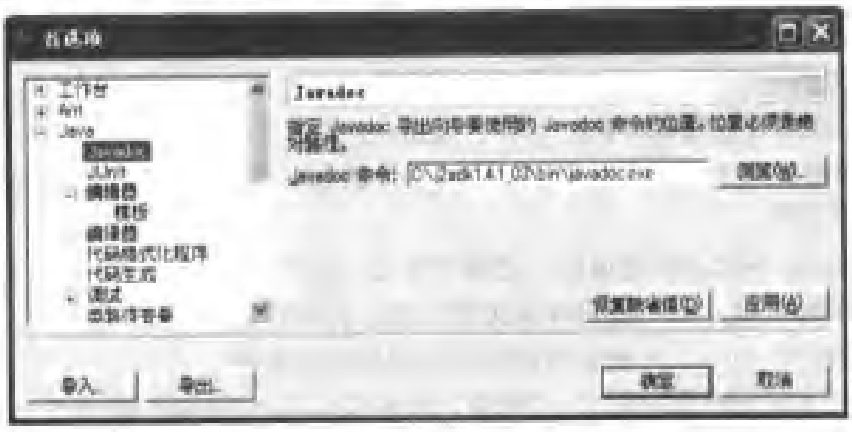


图 3-18 设置 Javadoc 命令

（2）使用“生成 Javadoc”向导。选择想要为其生成 Javadoc 文档的包、源文件夹或项目的集合（包含一个或多个元素）。可以从选择文件的上下文弹出菜单中选择“导出（Q）”，或者从菜单栏中选择“文件（F）”→“导出（Q）”来打开“生成 Javadoc”的向导，在打开的对话框中，从列表中选择“Javadoc”，并按“下一步”。

在图 3-19 的树控件中，选择想要为其生成 Javadoc 的元素，图中各选项含义如下。



图 3-19 “生成 Javadoc”向导

- 使用“为可视性成员创建 Javadoc”下面所列示的单选按钮来选择可视性。
- 使“使用标准 Doclet（S）”单选按钮处于选择状态。
- 在“目标（D）”字段中指定生成的 Javadoc 的位置。

按“完成（F）”为所选择的元素创建生成 Javadoc，或者按“下一步（N）”以指定做更多设定。

（3）为标准 doclet 配置 Javadoc 自变量。在上图中选择“下一步”，在打开的页面（见图 3-20）中配置 Javadoc 自变量：

① 通过选择在“基本选项”下面所列示的复选框来指定 Javadoc 选项；

② 通过选择在“说明这些标记组”中的复选框来更改将说明的标记；

③ 如果想要引用一个库中的类，并且将该库链接至它的 Javadoc，则在“选择 Javadoc 应该链接的引用类（R）：”的列表中选择该库，并按“配置（F）”来指定该库的 Javadoc 的位置；

④ 按“完成”以生成 Javadoc，或者按“下一步”来指定其他 Javadoc 生成选项。



图 3-20　“生成 Javadoc”向导

（4）配置 Javadoc 自变量。可以在文本区域中设置额外的 Javadoc 选项。在图 3-21 中，用户可以设置以下内容：

① 选择“将此 Javadoc 导出的设置另存为 Ant 脚本（S）：”复选框；

② 指定 Ant 脚本的位置；

③ 选择“在浏览器中打开生成的索引文件（P）”复选框。

按“完成”以生成 Javadoc。

Javadoc 命令生成的输出（包括错误和警告）将会显示在“控制台”视图中。

2. 查看 Javadoc 文档

（1）显示 Javadoc 信息。用户将光标悬浮于编辑器中的各个元素（如类名、方法名等）上，可以查看 Javadoc 帮助。如果系统提供了 Javadoc 信息，则会打开一个弹出窗口，显示 Javadoc 信息。查看 Javadoc 信息的方法有以下三种：

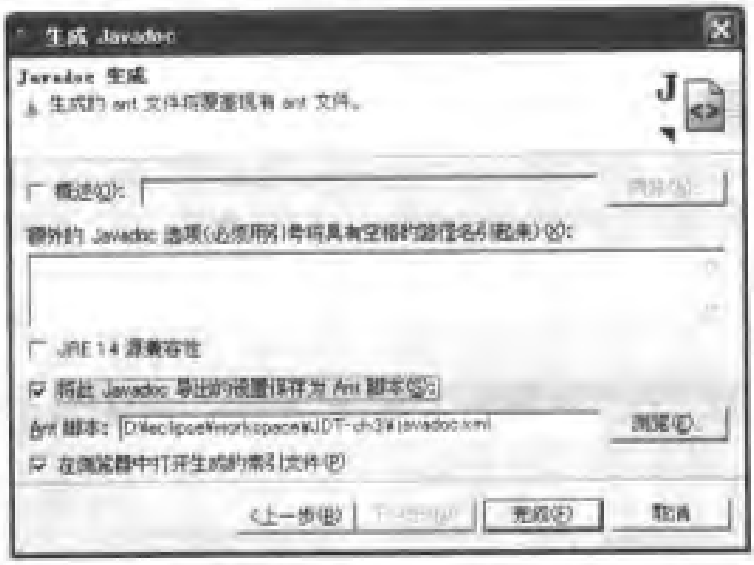


图 3-21 配置 Javadoc 自变量

- 将鼠标光标悬浮于想要查看的元素上；
- 通过菜单栏“编辑（E）”→“显示工具提示描述（T）”；
- 按“F2”键。

例如图 3-22 中光标悬浮于“set”上时，就可以查看到该方法是属于哪个类，以及该方法所带的参数等内容。



图 3-22 显示 Javadoc 信息

（2）用外部浏览器查看 Javadoc。还可以在外部浏览器中查看 Javadoc，在 Java 编辑器中将插入符号放在想要查看其 Javadoc 信息（例如，方法名）的元素上。按“Shift+F2”快捷键，或者从菜单栏中选择“浏览”→“打开外部 Javadoc”。

3.2.6 标识代码指示符

在这里，我们将详细介绍用于标识代码中的问题的不同指示符。通过在源代码中删除导入声明末尾的分号来添加语法错误。单击“保存”按钮后，用户将看到以下五种方

法来指示问题，如图 3-23 所示。

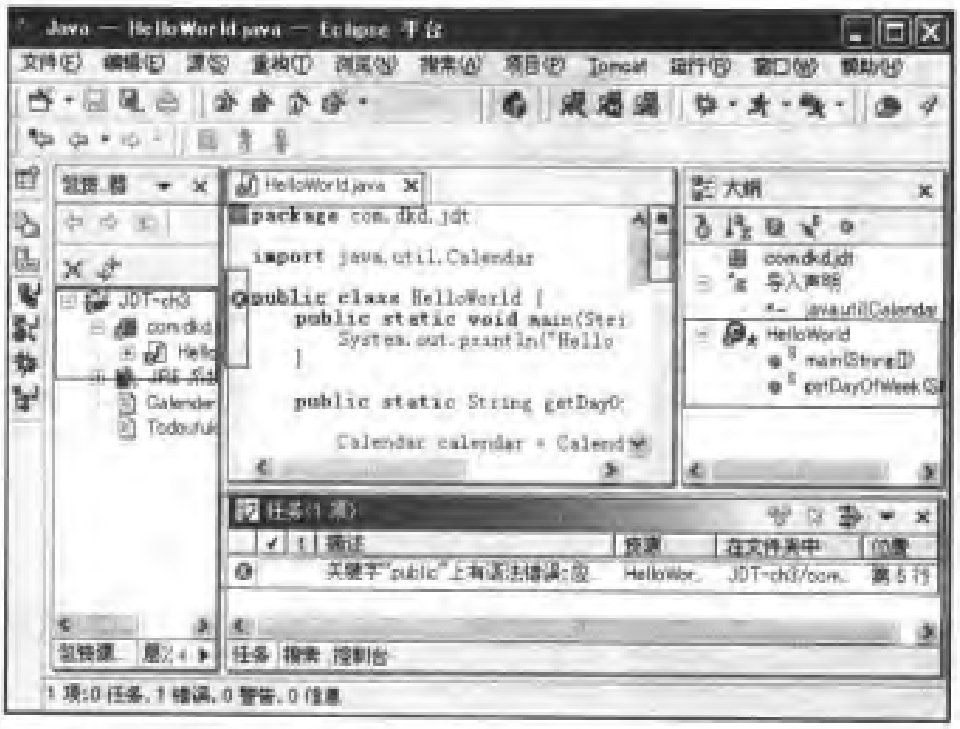


图 3-23　问题代码标识符

（1）在“任务”视图中，将列示问题，包括错误描述、资源、所在文件夹及文件的位置等。

（2）在“包资源管理器”视图、“类型层次结构”或“大纲”视图中，问题记号出现在受影响的 Java 元素及其父元素上，直至整个项目。

（3）在编辑器的垂直标尺中，问题标记显示在受影响的行附近。

（4）波浪线出现在可能导致错误的词语下面。

（5）问题标记对编辑器选项卡进行注释。

可以将鼠标指针悬浮在垂直标尺中的问题标记上来查看问题的描述，如图 3-24 所示。

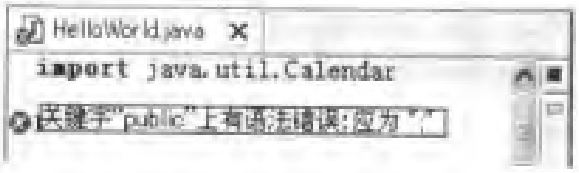


图 3-24　查看问题的描述

在“任务”视图中，选中列表中的问题，右击鼠标打开它的上下文菜单并选择“转至（G）”，文件将打开于编辑器中的问题位置处，如图 3-25 所示。此功能对于有多个类的项目来说，错误查找起来非常直接。



图 3-25 “任务”视图中的问题说明

在 getDayOfWeek(String strYMD)方法的最后一行上，将返回的变量 strReturn 更改为 strRetun。这时，问题突出显示下划线会出现在 strRetun 上指示发生了问题。将光标悬浮在突出显示问题上将会显示问题的描述。在标记栏上，一个灯泡形的问题标记出现。单击该问题标识可以进行快速修正，如图 3-26 所示。

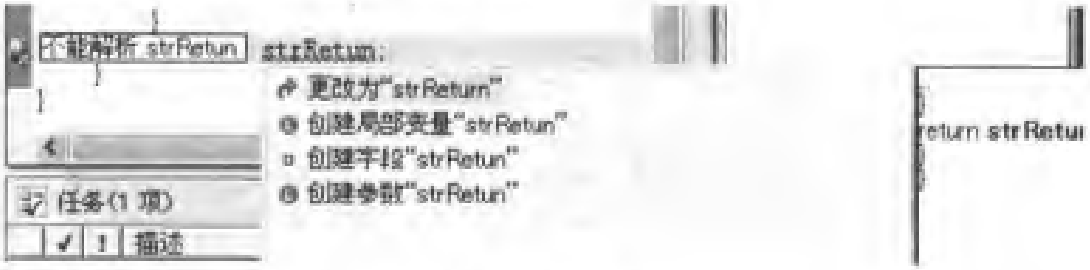


图 3-26 问题快速修正

3.2.7 格式化 Java 代码

1. 格式化文件或代码

JDT 的编辑器提供格式化 Java 代码的功能，打开一个 Java 文件，并选择想要格式化的代码。如果未选择任何内容，则格式化所有编辑器内容。可以通过下列任意一种方式来格式化代码。

- 在编辑器中按鼠标右键的弹出菜单中选择“源（S）”→“格式化（O）”；
- 按“Ctrl+Shift+F”快捷键；
- 从菜单栏中选择“源（S）”→“格式化（O）”。

2. 设置代码格式化首选项

用户可以根据个人需要来设置代码的格式，通过使用“代码格式化”首选项页面来指定用于代码格式化的首选项。方法是：从工作台菜单栏中，选择“窗口”→“首选项”→“Java”→“代码格式化程序”，打开“代码格式化程序”首选项页面，如图 3-27 所示。在“新行”、“行分割”和“样式”三个选项卡中，选择想要格式化程序遵循的代码格式化约定。在该页面下面的窗口中，还可以观察每个代码格式化选项的示范效果，并查看已格式化的代码的预览效果。完成时单击“确定”。



图 3-27　代码格式化窗口

3.2.8　使用模板编程

模板是对源代码中重复出现的编码模式的结构化描述，为了避免输入完整的代码段，为用户提供了一些常用的模板，例如，使用包含数组下标的 for 循环来对数组元素进行迭代。Java 编辑器支持使用模板来填充常用的源模式，已经定义了许多公共模板。读者可以在“窗口”→“首选项”→“Java”→“编辑器”→“模板”中浏览它们，用户还可以根据需要创建自己的模板或编辑现有模板。

1．模板的使用

模板可以具有一些变量，这些变量是模板模式的动态部分的占位符。当将模板插入到 Java 编辑器中并且模板模式包含了模板变量时，编辑器就进入模板编辑方式。编辑器在第一个变量下面自动添加了下划线，并且选择了该变量。可以通过在编辑器中输入新的变量名来修改该变量名。如果模板模式中多次存在同一个变量名，在处于模板编辑方式下更新一个变量名的同时，其他所有出现该变量名的地方都将随之更新。通过按 Tab 键导航至下一个惟一模板变量，按 Shift+Tab 导航至上一个惟一模板变量。通过在最后一个模板变量上按 Tab 键或者按 Esc 或 Enter 键来脱离模板编辑方式。

下例说明如何使用编辑器提供的公共模板，添加如下所示的新方法。

```
public void getValues() {
    Integer[] expected= new Integer[3];
    for
```

将插入记号定位在想要插入模板的位置，即将光标置于 for 末尾，按“Ctrl+空格键”以启用内容辅助。将会弹出“for”循环的公共模板的列表，如图 3-28 所示。当选中一个模板上时，用户可以在模板的帮助消息中看到所选模板的代码预览。

选择“for-iterate over collection”条目并按 Enter 键以确认该模板，这时在源代码中插入该模板。如果我们想更改下标变量（即变量下面有浅绿色短横线）的名称，例如将“i”更改为“e”。为此，只需简单地在“i”的位置上输入“e”，这样就自动更改了下标变量，并且，在所有位置的下标变量的名称都会自动更改。这是因为插入模板时，所有同名的模板变量都会彼此连接。所以，更改一个名称也将更改所有其他名称，如图 3-29 圆圈中所示。

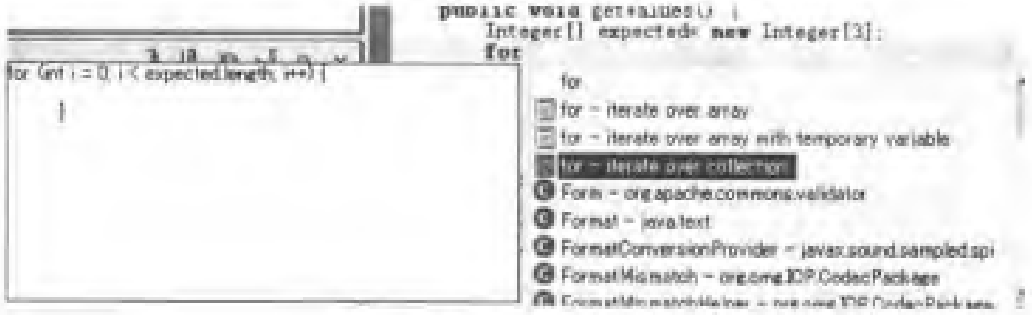


图 3-28 模板预览

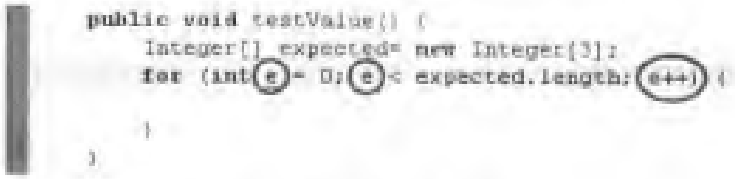


图 3-29 模板下标变量

按 Tab 键将把光标移至代码模板的下一个变量。这是 expected 数组，我们可以不用更改其名称（因为模板正确地猜出了名称），再次按 Tab 键，由于没有剩下要编辑的变量，这时就会离开该模板。如图 3-30 所示。

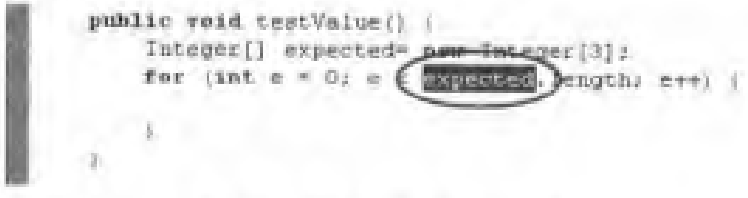


图 3-30 模板自动更新下标变量

2. 编写自己的模板

用户可以根据需要，编写和定义自己的模板。具体方法可以参照以下步骤进行。

（1）进入首选项窗口：“窗口”→“首选项”→“Java”→“编辑器”→“模板”，并按“新建”按钮。这时，打开一个“新建模板”的窗口，如图 3-31 所示。



图 3-31　新建模板窗口

（2）在“名称（N）”字段中，输入模板的名称。此名称不需要是惟一的，它可用于从“代码辅助”列表中选择模板。

（3）使用“上下文（C）”组合框来指定模板的上下文：

① 如果要在正常的 Java 代码中使用模板，则选择“java”。

② 如果要在 Javadoc 注释中使用模板，则选择“javadoc”。

（4）在“描述（D）”字段中，输入对模板的简短描述。

（5）在“模式（P）”文本字段中输入模板模式，模式可以包含预定义的和定制的模板变量。

模板变量是模板模式的动态部分的占位符（即，它们在特定模板的每个应用程序中是不同的）。在插入模板之前，要对模板模式中的变量进行评估，并将占位符替换为评估值。

插入模板变量要注意以下几点：

- 变量的格式为${variable_name}。如果想要显示美元符号$，则必须使用两个美元符号$$，或者使用变量${dollar}来对它进行转义。
- 要插入预定义的模板变量，使用“插入变量”按钮，或者按“Ctrl+空格键”，并从弹出的显示列表中选择变量，如图 3-32 所示。

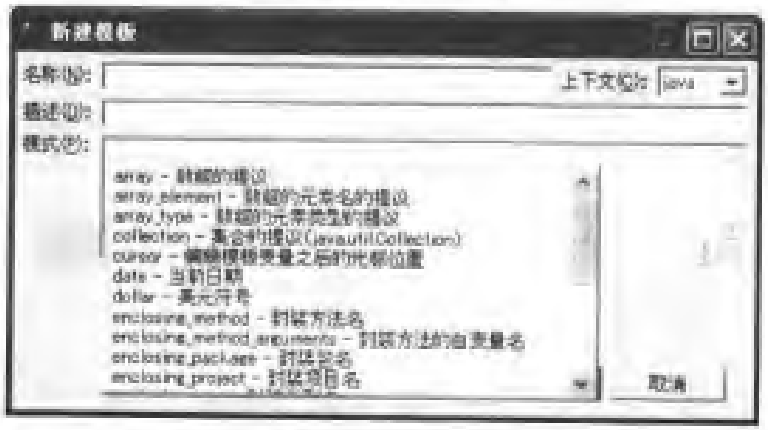


图 3-32　插入模板变量

- 如果想插入自定义的模板变量，需要将该模板变量求值为变量本身的名称，而且必须确保该名称与特定上下文中预定义的模板变量名不发生冲突。

3.2.9 使用本地历史记录

JDT 以如下三种方式扩展本地历史记录：

- 不仅可以在“导航器”视图中，而且可以在“包资源管理器”视图中将文件替换为本地历史记录中的一个版本。
- JDT 允许将个别 Java 元素（类型及其成员）与本地历史记录中的版本进行比较，或替换为本地历史记录中的版本。
- JDT 允许恢复从工作台中删除的已经保存在本地历史记录中的 Java 元素和文件。

所谓版本就是，文件和 Java 元素（例如，类型及其成员）会经常被更改，它们的外观在某个时间点的“快照”（保存在本地历史记录中）就称为一个版本。

1. 将 Java 元素与本地历史记录版本进行比较

（1）首先要确保 Java 视图可视。打开想要与本地历史记录中的版本进行比较的 Java 文件，配置“包资源管理器”来显示或不显示文件中的 Java 元素。使用“窗口”→“首选项”→“Java”→“编辑器”→“外观”，来设置显示“包资源管理器”中成员的首选项。

（2）选择想要比较的元素。选中要比较的元素，例如，getValues()，右击鼠标，从元素的弹出上下文菜单中，选择“比较对象”→“本地历史记录中的元素”。在得到的对话框的顶部窗格中，会显示本地历史记录中所选元素的所有可用版本，如图 3-33 所示。在顶部窗格中选择一个版本，以查看所选版本与工作台中的版本之间的差异。“Java 源代码比较”窗格显示所选历史记录资源与现有工作台资源之间差异的详细信息。

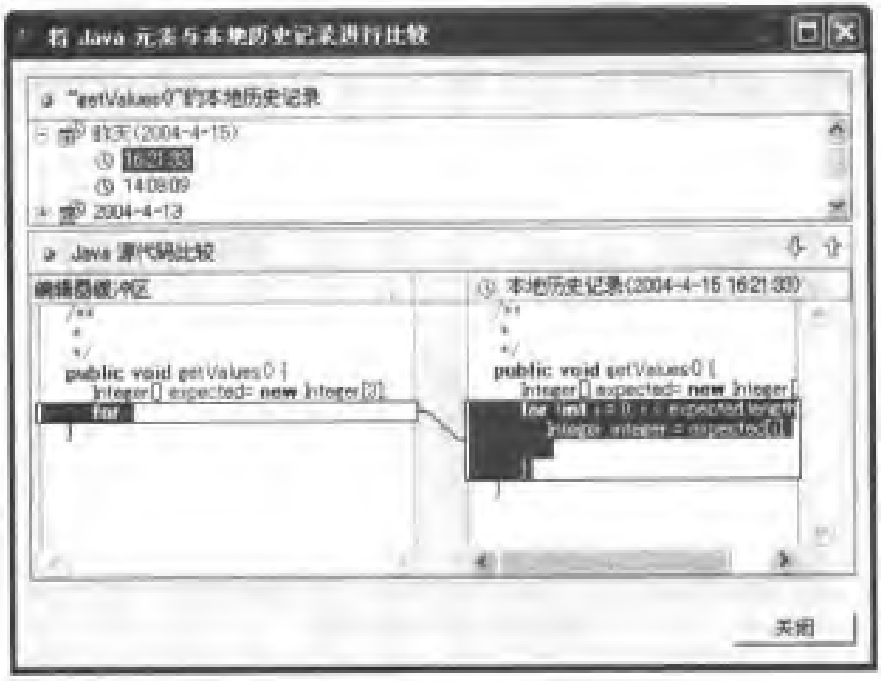


图 3-33 将 Java 元素与本地历史记录进行比较

2. 将 Java 元素替换为本地历史记录版本

选中要比较的元素，例如，getValues()，右击鼠标从元素的弹出上下文菜单中，选择“替换为”→“本地历史记录中的元素”。在得到的对话框的顶部窗格中，会显示本地历史记录中所选元素的所有可用版本。在顶部窗格中选择一个版本，以查看所选版本与工作台中的版本之间的差异。当用户找到想要替换现有 Java 元素的版本时，单击“替换”按钮。本地历史记录版本将替换编辑器中的当前版本。注意：此时尚未保存已更改的编译单元。确认之后，按“Ctrl+S”组合键保存文件。

3. 恢复已删除的工作台元素

我们先将 getValues()方法删掉，如果想恢复已删除的方法，也可以从本地历史记录中将它恢复。具体做法是：

在 Java 视图中，选择想要将已删除的元素恢复为其容器类型的元素。在本例中，选中 RenameTest。从上下文菜单中，选择“从本地历史记录恢复”。在获得的如图 3-34 所示的对话框的左上方窗格中，会显示本地历史记录中所选元素的所有可用版本。在右上窗格中，选择想要替换的所有元素。对于选择的每个元素，在右窗格中选择一个版本，并在“本地历史记录”窗格中查看保存在本地历史记录中的版本。当已经确认了想要恢复的版本时，按“恢复”按钮，这就将本地历史记录版本装入到编辑器中了。

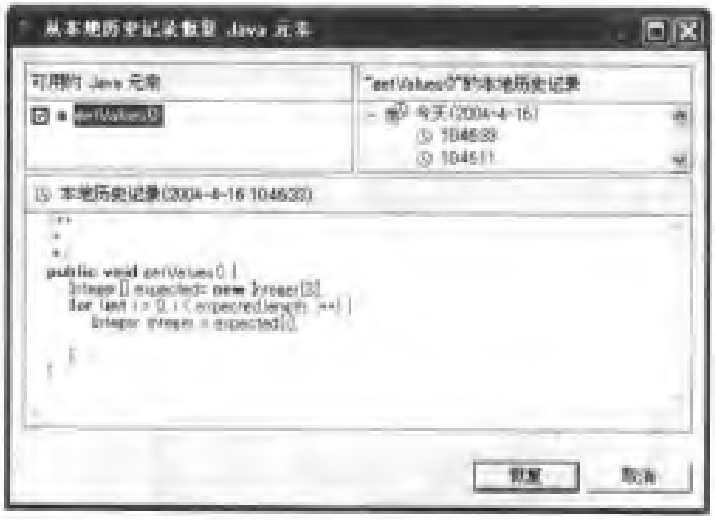


图 3-34　从本地历史记录恢复 Java 元素

3.3　JDT 的相关视图

3.3.1　“包资源管理器”视图

“包资源管理器”视图显示工作台中的 Java 项目的 Java 元素层次结构。它提供“导航器”中显示特定于 Java 视图的资源。元素层次结构是从项目的构建类路径中派生出来的。对于每个项目，其源文件夹和引用的库都显示在树中，可以打开和浏览内部和外部

JAR 文件的内容。

1. 使用“包资源管理器”来浏览 Java 元素

在如图 3-35 所示的“包资源管理器”视图中，展开“JDT-ch3”项目可以查看它的包，展开“com.dkd.jdt”包可以查看到包含在该包中的 Java 文件，展开 Java 文件“HelloWorld.java”，可以显示源代码文件中的 Java 子元素，导入声明、公共类型和它的成员（字段和方法）显示在树中。



图 3-35 使用“包资源管理器”来浏览 Java 元素

2. 在“包资源管理器”视图中打开类型

可以对项目的类路径上包括的任何类型打开“包资源管理器”。从菜单栏中，通过选择“浏览”→“转至”→“类型”来打开“转至类型”对话框，如图 3-36 所示。在“选择类型（C）”字段中，输入表达式来缩小可用类型的列表，需要时使用通配符（*）。输入时，会过滤该列表以便只显示与当前表达式匹配的类型。例如，输入“*.java”，在列表中只会显示 Java 类型的文件。在“匹配类型（M）”列表中，选择一种类型。当一种类型具有多个定义时，在“限定符（Q）”列表中选择一个包。完成上述操作时单击“确定”。所选类型便显示在“包资源管理器”中。



图 3-36 “转至类型”对话框

注意：如果应用了“包资源管理器”过滤器，则可能不会显示。

3. 过滤元素

在“包资源管理器”工具栏上，单击“菜单”→“过滤器（F）”，如图 3-37 所示。



图 3-37　过滤元素

在弹出的如图 3-38 所示的对话框中，选择对话框顶部的“名称过滤器模式（N）”的复选框，在下面的文本字段中指定期望的模式，将会隐藏相匹配的名称。例如，在文本框中输入了“*.js”，则在图 3-37 的视图中所有 JS 文件都将隐藏起来。

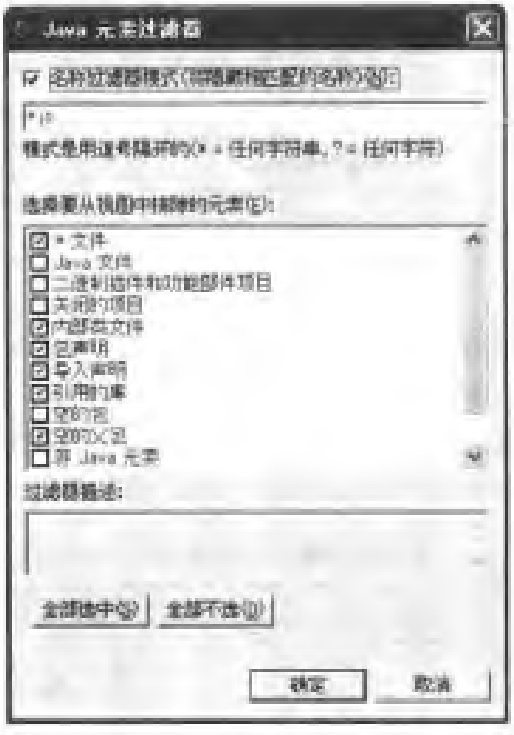


图 3-38　“Java 元素过滤器”对话框

4. 显示和隐藏元素

在图 3-38 所示中，在“选择要从视图中排除的元素（E）”的列表中，选择某个元素的复选框，则该类型的元素就会在视图中隐藏起来，反之，清除某个元素的复选框，

则该类型的元素就会在视图中显示出来。

5. 移动文件夹、包和文件

在“包资源管理器”中，选择想要移动的文件夹、包或文件，将所选资源拖放到期望的位置。

注意：可以使用拖放操作在不同的工作台窗口之间移动资源。选择想要移动的资源，并将它拖放到期望的目标中。如果想要进行复制而不是移动，则在拖动时应按住“Ctrl”键。还可以使用拖放操作来在工作台与桌面之间（双向）复制资源，但是，不能使用拖放操作来将文件从桌面移到工作台中。

3.3.2 “大纲”视图

“大纲”视图与编辑器相关联，它显示活动 Java 编译单元的结构，包括包声明、导入声明、字段、类型和方法等，它是在用户编辑编译单元时实时更新的。“大纲”视图使用图标来对 Java 元素进行注释。图标指示 Java 元素是静态的（static）、抽象的（abstract）还是终态的（finally）。例如，图 3-39 中所示的 main()方法的图标所指示的是静态的 Java 元素。



图 3-39 大纲视图

Java 编辑器可以配置为显示整个编译单元或者只显示单个 Java 元素。按下工具栏中的“仅显示所选元素的源”的按钮，选择“大纲”视图中某个方法，如图 3-40 所示，选择 main(String[])方法，这时在编辑器中只显示了 main（String[]）方法的源代码。再次按“仅显示所选元素的源”按钮以再次查看整个 Java 文件。

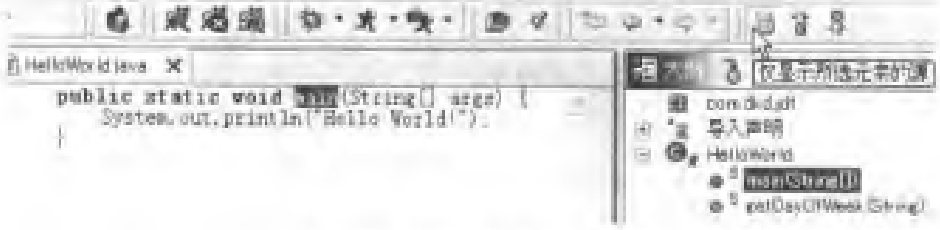


图 3-40 仅显示所选元素的源

对选择的元素打开编辑器可以在 Java 源代码编辑器或代码片段编辑测试窗中选择类型、方法或字段的名称，然后对元素的定义打开编辑器。

在Java编辑器中，选择类型、方法或字段的名称，可执行下列操作之一：

- 从菜单栏中，选择“浏览”→“打开声明”；
- 从编辑器的弹出菜单中，选择“打开声明”；
- 按“F3”键。

如果具有多个同名的定义，则会显示一个对话框，选择想要打开的一个定义。

3.3.3 “层次结构”视图

“层次结构”视图允许用户查看类型的完整层次结构，可以只查看它的子类型或者只查看它的超类型，如图3-41所示。

1. 对Java元素打开类型层次结构

在“包资源管理器”视图中选择Java元素，可以使用如下几种方法来打开类型层次结构：

- 按F4键；
- 右击鼠标，在弹出菜单中选择“打开类型层次结构”；
- 将元素拖放到“层次结构”视图；
- 按“Ctrl+Shift+H”键，并从获得的对话框中的列表中选择一种类型（这仅对类和接口起作用）；
- 从菜单栏中选择“浏览”→“打开层次结构中的类型”；
- 从类型层次结构查看器的弹出菜单中选择“聚焦于...(F)”。



图3-41　层次结构视图

2. 使用“层次结构”视图覆盖方法

可以使用“层次结构”视图来覆盖方法，打开想要在其中添加覆盖方法的类型的类型层次结构。注意，是显示子类型、超类型还是类型层次结构并不重要。在“层次结构”视图的列表窗格中，确保“显示所有继承的成员”按钮是活动的，以便查看所有继承的成员。注意，如果一个过滤器（例如，“隐藏字段”或“隐藏静态成员”）是活动的，则该选择可能不会显示所有成员。选择要覆盖的方法，按鼠标右键，选择在类名中覆盖（其

中，类名是将在其中创建方法的类的名称）。如图 3-42 所示，选择“在‘HelloWorld’中覆盖（O）”。



图 3-42 使用“层次结构”视图来覆盖方法

3. 查找被覆盖的方法

打开所选方法的声明类型的类型层次结构，打开“显示子类型层次结构”工具栏按钮。在“层次结构”视图的列表窗格中，确保打开了“锁定视图并显示层次结构中的成员”按钮。此选项锁定方法窗格中的当前类，并且只显示顶部视图中实现了已锁定类的当前所选方法的那些类，如图 3-43 所示。

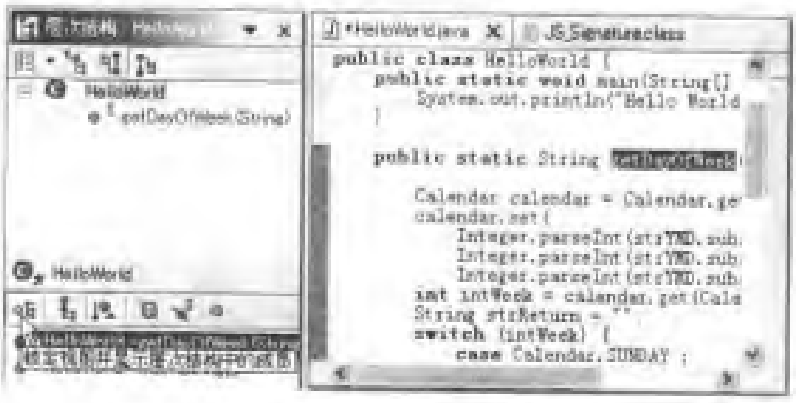


图 3-43 查找被覆盖的方法

3.3.4 “Java 浏览”透视图

“Java 浏览”透视图用来浏览 Java 项目结构，它包含一个编辑器和四个视图，如图 3-44 所示。

1. “项目”视图

“项目”视图显示 Java 项目、源文件夹、外部库和内部库。注意，在此视图中存在的源文件夹和库（无论是内部的和外部的）是不可扩展的。当选择它们时，它们的内容就会显示在“包”视图中。

2. “包”视图

“包”视图显示当前选择的 Java 项目、源文件夹或库中的 Java 包列表。通过选择“项目”视图中的项目来显示该项目中所包含的包。

3. “类型”视图

“类型”视图显示当前选择的包中的 Java 类型的列表。通过选择“包”视图中的包，在“类型“视图中显示所选包中的 Java 类型的列表。



图 3-44　“Java 浏览”透视图

4. “成员”视图

“成员”视图显示类型、编译单元或 CLASS 文件的内容。通过选择“类型”视图中的类型，在“成员“视图中显示所选类型中的方法等。

3.4　运行与调试

从工作台中启动 Java 程序，可以通过两种方式来启动程序，即运行方式和调试方式。在运行方式下，程序直接被执行，不会暂挂或检查执行；在调试方式下，可暂挂执行和继续执行，并且可检查变量，以及对表达式进行求值。

3.4.1　运行

修改完“HelloWorld.java”程序，可单击“保存”按钮或者按“Ctrl+S”键保存程序，Eclipse 将自动编译“HelloWorld.java”，并在“资源”透视图中的 JDT-ch3 项目下自动生成了“HelloWorld.class”，如图 3-45 所示。

1. 启动 Java 应用程序

启动 Java 程序的最简单方法是以 Java 应用程序方式运行它。在“大纲”视图中，选择想要启动的 Java 编译单元或类文件，如图 3-46 所示。选择 HelloWorld 类，它旁边的图标表示该类定义了 main 方法。选择菜单栏中的“运行（R）”→“运行为（S）”→“Java 应用程序”或者选择工具栏中的“运行（）”按钮旁的三角按钮，在下拉菜单中选择“运行为（S）”→“Java 应用程序”。这样就启动了程序，并且在“控制台”中显示结果输出，如图 3-47 所示。



图 3-45 生成 class 类



图 3-46 定义了 main 方法的类

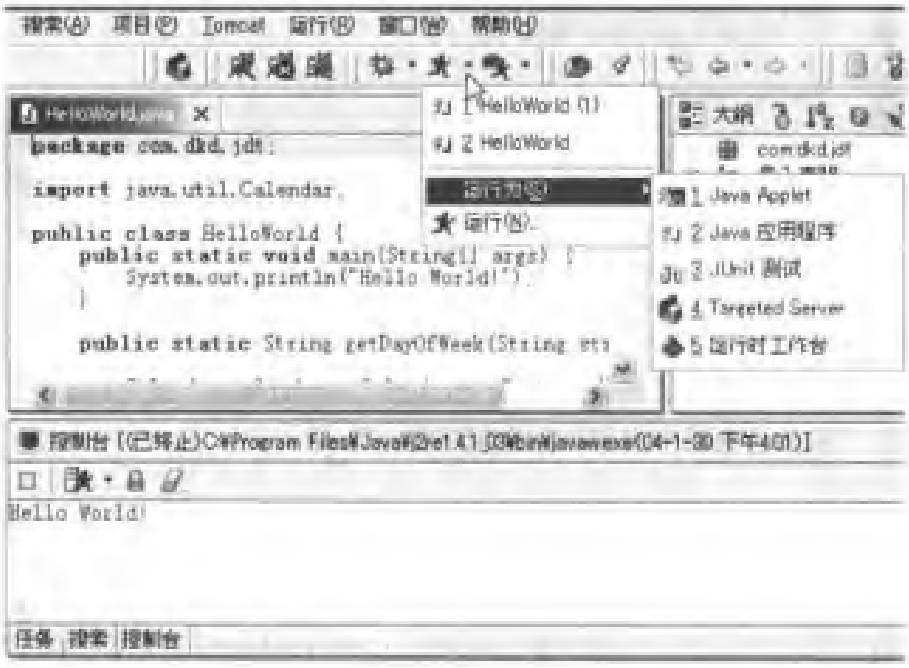


图 3-47 控制台

如果 Java 程序被构造为 applet，则可以选择“Java Applet”方式来启动程序。

2. 创建 Java 应用程序启动配置

当选择“运行（R）”→“运行为（S）”→“Java 应用程序”来启动类时，实际上是在使用类属“Java 应用程序”启动配置运行类，该启动配置从 Java 项目和工作台首选项中派生大多数启动参数。在某些情况下，用户可能会需要重设派生的参数或者指定附加自变量，通过用户创建自己的 Java 应用程序启动配置来实现此目的，具体方法如下。

（1）从工作台菜单栏中选择“运行（R）”→“运行（N）”或者选择“运行（R）”→“调试（B）”来打开一个对话框，如图 3-48 所示，在该对话框中用户可以创建、修改和删除不同类型的启动配置。



图 3-48　运行窗口的主要选项卡

（2）在启动配置类型的左边列表中选择 Java 应用程序并按“新建（W）”。这将为 Java 应用程序创建新的启动配置。右边的选项卡允许用户控制启动的特定方面。下面介绍各选项卡的定义：

- “主要”选项卡定义要启动的类。在“项目（P）”字段中输入包含要启动的类的项目名称，并在“Main 类”字段中输入主类的全限定名称。如果想要每次以调试方式启动程序时，程序在 main 方法中停止的话，则选择“在 main 中停止”复选框。

注意：用户不是必须指定项目，但是指定了项目将允许选择缺省的类路径、源查找路径以及缺省的 JRE。

- “自变量”选项卡定义要传送至应用程序和虚拟机（如有的话）的自变量。还可以指定由已启动的应用程序使用的工作目录。
- “JRE”选项卡定义要用来运行或调试应用程序的 JRE。可以从已经定义的多个 JRE 中选择一个 JRE，也可以定义新的 JRE。
- “类路径”选项卡定义运行或调试应用程序时使用的类文件的位置。缺省情况下，用户类（L）和引导程序类（B）位置是从相关联的项目的构建路径派生的，用户也可以在此处覆盖这些设置。
- “源代码”选项卡定义调试 Java 应用程序时用来显示源的源文件的位置。缺省情

况下，这些设置是从相关联的项目的构建路径派生的，用户也可以在此处覆盖这些设置。

- “公共”选项卡定义有关启动配置的一般信息。可以选择将启动配置存储在特定文件中，并指定当启动了启动配置时哪些透视图将变成活动的。

（3）设置执行自变量。如果想要指定程序的执行自变量，则必须定义指定自变量的启动配置。在图 3-48 的运行配置对话框中，选择“自变量”选项卡，如图 3-49 所示。



图 3-49　运行窗口的自变量选项卡

可以根据需要来指定下列字段：

- 程序自变量（A）——在这里可以设定 main()方法的参数。
- VM 自变量（G）——在这里可以更改 JVM 的值。例如，用户可能需要告诉 VM 是否使用 JIT 编译器，或者用户可能需要指定 VM 应该使用的最大堆栈的大小。
- 工作目录——用于已启动的进程的工作目录。如果不想使用缺省工作目录，则取消“使用缺省工作目录（K）”的复选框，并指定要用作已启动进程的工作目录的“工作区（P）”或“本地目录（L）”。

在完成所有设置后单击“应用”或“关闭”按钮。每当启动此配置时，就将使用这些执行自变量。

（4）重新启动程序。工作台保留已启动和已调试的每个程序的历史记录。要重新启动程序，执行下列操作之一：

- 从工具栏中的“运行”或“调试”按钮下拉菜单中选择上一次启动。

- 从菜单栏中，选择“运行（R）”→“运行历史记录（U）”，或者选择“运行（R）”→“调试历史记录（H）”，并从这些子菜单中选择上一次启动，如图 3-50 所示。
- 在“调试”视图中，选择想要重新启动的进程，右击鼠标，并从该进程的上下文菜单中选择“重新启动（L）”。

要重新启动最近一次的启动，可采用下列方式之一：

- 单击“运行”或“调试”按钮（而不使用按钮下拉菜单）。
- 从工作台菜单栏中选择“运行（R）”→“运行上次启动（R）”（或按“Ctrl+F11”快捷键），“运行（R）”→“调试上次启动（D）”（或按“F11”快捷键）。参见图 3-50。

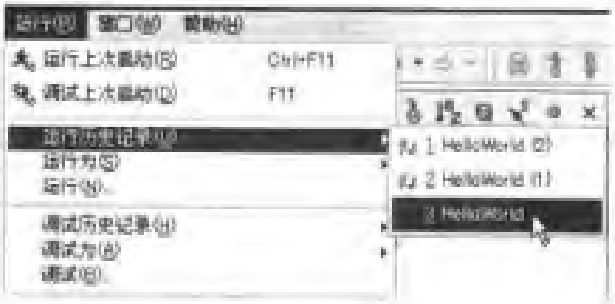


图 3-50　运行历史记录

3.4.2　JDT 调试器

JDT 中包含一个调试器，它允许用户检测和诊断在本地运行或远程运行的程序中的错误。调试器允许通过设置断点、暂挂已启动的程序、单步执行代码和检查变量的内容来控制程序的执行。

调试器采用的是客户机 / 服务器设计，因此，用户可以调试在网络中的其他系统上远程运行的程序，也可以调试在本地工作站上运行的程序。调试客户机在工作站上的工作台中运行，调试器服务器与要调试的程序在同一个系统上运行，这可以是在工作站上启动的程序（本地调试），也可以是在可以通过网络访问的计算机上启动的程序（远程调试）。

可以通过定制调试器来控制启动时所显示的活动透视图，通过这个定制可以在以调试方式或运行方式启动程序时，控制将设置哪个透视图变为活动的。以调试方式或运行方式启动程序时，执行下列操作激活特定透视图：

（1）通过菜单栏上的“窗口”→“首选项”→“调试”打开调试器首选项页面，如图 3-51 所示。

（2）从“用于调试的缺省透视图（P）”下拉框中选择一个透视图。按“应用（A）”→“确定”保存后，每当以调试方式启动程序时，用户选择的透视图就会变成活动的。

（3）从“用于运行的缺省透视图（T）”下拉框中设置透视图。按“应用（A）”→“确定”保存后，每当以运行方式启动程序时，用户选择的透视图就会变成活动的。



图 3-51 调试首选项窗口

（4）要覆盖对特定启动配置的缺省选项，方法是在“启动配置”对话框中选择“公共（C）”选项卡，在“启动时要切换至或打开的透视图”选项中选择一个透视图。如果以运行方式或调试方式启动程序时未打开所选择的透视图，则系统将创建该透视图。

3.4.3 断点

断点是调试程序的重要手段，断点可导致程序线程的执行到设置断点的位置时暂挂。用户可以通过“断点”视图中的上下文菜单来启用和禁用断点。断点显示在垂直编辑器标尺和“断点”视图中。

启用断点后，它将导致线程在每当遇到断点时暂挂。已启用的断点是用蓝色圆圈来指示的。在 VM 装入已启用的断点的类并且成功安装断点后，这些断点显示为带有复选标记覆盖层。禁用断点后，它将不会导致线程暂挂，禁用的断点是用白色圆圈来指示的。

1. 添加行断点

行断点是在程序的可执行上设置的。在编辑器区域中，打开想要添加该断点的文件，将光标置于想要添加断点的那一行代码正左方的标记栏（垂直标尺）上，右击鼠标弹出上下文菜单，并选择“添加断点（B）”，如图 3-52 所示。或者可以双击源代码行旁边的标记栏，新的断点标记出现在标记栏上，就在添加了断点的那一行的正左方。而且，新断点还出现在“断点”视图列表中（见图 3-53）。

启用断点时，线程执行会在在执行该代码行之前暂挂。调试器选择已暂挂的线程，并显示在该线程的堆栈上的堆栈帧。设置了断点的行在“调试”透视图的编辑器中突出

显示。



图 3-52　断点标识



图 3-53　断点视图

2. 除去行断点

当不再需要断点时，可以很容易地除去它们。除去断点的方式有多种，可以使用下列任意一种方式：

- 在想要除去断点的那一行的正左方，右击鼠标打开标记栏弹出菜单，并选择“除去断点（B）”。
- 直接双击断点图标来除去断点。
- 在“断点“视图中除去断点，选择要除去的断点，并从上下文菜单中选择“除去（Q）”，也可以使用“全部除去（L）”操作除去所有断点。

3. 启用和禁用断点

如果是经常在同一位置添加和除去断点，则可以考虑在不需要断点时禁用它，并且在再次需要该断点时又重新启用它。

要禁用断点可采用下列任意一种方式：

- 在编辑器的标记栏中打开该断点的上下文菜单，并选择“禁用断点（D）”。
- 在“断点”视图中找到该断点，打开该断点的上下文菜单，并选择“禁用（D）”。

此时断点的图像将更改为白色圆圈。

要启用断点可采用下列任意一种方式：

- 在编辑器的标记栏中找到该断点，右击鼠标打开该断点的上下文菜单，并选择“启用断点（E）”。
- 在“断点”视图中找到该断点，打开该断点的上下文菜单，并选择“启用（D）”。

此时断点的图像将恢复为蓝色圆圈。

4. 设置方法断点

当想对整个方法设置断点时就要使用方法断点。在“大纲”视图中打开类，并选择想要在其中添加方法断点的方法。从方法的弹出菜单中，选择“添加 / 除去方法断点（M）”。断点便出现在“断点”视图及所选方法的文件编辑器的标记栏中，如图 3-54 所示。启用断点时，如果输入了方法，线程执行会在执行该方法中的任何行之前暂挂。还可以将方法断点设置为在方法退出时中断。在“断点”视图中选择断点，并选择其上下文菜单中的“退出（X）”项。方法断点可以像行断点一样除去、启用和禁用方法断点。图 3-54 中所举的例子是在 main（String[]）方法退出时设置的方法断点。



图 3-54　设置断点

5. 命中计数

命中计数可以应用于行断点、异常断点、监视点和方法断点。将命中计数应用于断点时，当第 n 次遇到断点时，断点将暂挂线程的执行，直到重新启用该断点或者更改或禁用命中计数为止。

对断点设置命中计数的方法是：选择要对其添加命中计数的断点，从断点的弹出菜单中选择“命中计数（H）”。在“设置断点命中计数”的对话框中，如图 3-55 所示，在“输入断点的新命中计数（E）”字段中，输入想要在暂挂执行之前遇到断点的次数。



图 3-55　设置断点命中计数

例如，在“输入断点的新命中计数（E）”字段中输入 2，则当第二次遇到断点时，就会暂挂遇到断点的线程。如图 3-56 所示的调试结果，在第二次运行到 System.out.println("break point 2")时的行断点时线程暂挂。

6. 管理条件断点

启用条件可以应用于行断点，以便断点在满足启用条件时或者更改启用条件时执行暂挂线程。选择要设置条件的断点，在上下文菜单中选择“断点属性”，在打开的“Java 行断点属性”对话框中对断点进行条件设置。图 3-57 所示为“断点属性”对话框。在属

性对话框中，选择“启用条件（N）”复选框，在“条件（C）”字段中，输入断点条件的表达式，可以按“Ctrl+空格键”来获取代码辅助。



图 3-56　调试结果



图 3-57　“Java 行断点属性”对话框

如果想要每当条件求值为 true 时断点就停止，则暂挂就选择“条件为 true”选项。所提供的表达式必须是布尔表型达式。如果想要仅当条件的结果更改时断点才停止，则选择“条件的值更改”选项。单击“确定”以关闭该对话框并提交更改。启用断点时，

如果断点条件等于 true，则线程执行会在执行该代码行之前暂挂。条件断点在断点图标上有问号覆盖层。

7. Java 异常断点

通过指定异常断点，可以在抛出异常时暂挂线程的执行。可以在未捕获异常或捕获异常的位置暂挂执行。从“断点”视图或工作台“运行”菜单中选择“添加 Java 异常断点”，就会显示一个列示了所有可能发生的异常的对话框。输入想要捕获的异常的名称，或者从列表中选择异常的名称。在页面的底部，使用复选框来指定想要在抛出异常的位置如何暂挂执行。如果想要在抛出异常并且捕获异常的位置暂挂执行，则选择“已捕获（C）”，如果想要在未捕获异常的位置暂挂执行，则选择“取消捕获（U）”。Java 异常断点具备与常规断点一样的属性，比如启用和禁用异常断点以及具有命中计数等。

3.4.4 本地调试

本地调试是最简单并且最常用的一种调试。完成编辑和构建 Java 程序后，可以通过选择菜单中的“运行”→“调试（B）...”项在工作站上启动程序。以此方式启动程序将在调试器客户机与正在启动的 Java 程序之间建立连接。然后，通过使用断点、单步执行或表达式求值等方式来调试程序。

注意：为了便于设置行断点或单步执行，在编写程序时应该尽可能不要将多个语句放在同一行上，这是因为某些调试器功能部件是以行为基础来运行的。

1. 以调试方式启动 Java 程序

通过调试方式启动程序，可以使用调试器来暂挂和继续执行程序、检查变量和对表达式进行求值。用户可以通过下列任意一种方式以调试方式启动 Java 程序：

- 在“包资源管理器”中，选择想要启动的 Java 编译单元或类文件，在菜单上选择“运行（R）”→“调试为（A）”→“Java 应用程序”。
- 也可选择工具栏中的调试按钮（）上的下拉菜单中的“调试为（A）”→“Java 应用程序”。

启动程序后，工作台就进入到“调试”透视图中，如图 3-58 所示，并且已启动的进程显示在“调试”视图中，如图 3-58 中左上位置所示。

2. 暂挂线程

要暂挂正在执行的线程：在“调试”视图中选择该线程。单击“调试”视图工具栏中的“暂挂”按钮。该线程就会暂挂它的执行。将显示该线程的当前调用堆栈，并且在“调试”透视图中的编辑器中突出显示执行的当前行。线程暂挂时，会自动选择该线程的顶部堆栈帧。

“变量”视图显示堆栈帧的变量以及它们的值。可通过展开复杂变量以显示它们的成员的值来进一步检查复杂变量。当暂挂了线程并且将光标悬浮在 Java 编辑器中的一个变量上时，就会显示该变量的值。



图 3-58　"调试"透视图

3. 继续执行暂挂的线程

要继续执行暂挂的线程：在"调试"视图中选择该线程或者它的堆栈帧。单击"调试"视图工具栏中的"继续"按钮（或者按 F8 键）。线程将继续执行，但是将不再对该线程显示堆栈帧，同时也清除了"变量"视图。

4. 单步遍历 Java 程序的执行

如果想要程序在 main 方法中停止以便可以单步遍历它的整个执行，则创建"Java 应用程序"启动配置，并选择"主要"选项卡上的"在 main 中停止"复选框（参见图 3-48）。还可以通过选择项目而不是编译单元或类文件来调试 Java 程序。系统将提示用户从定义 main 方法的那些类中选择一个类。（如果在项目中只找到了一个具有 main 方法的类，则会像选择了该类一样而启动该类。）

暂挂线程时，可以使用单步控制来逐行单步完成程序的执行。如果在执行单步操作时遇到了断点，则在该断点处将暂挂执行，并且该单步操作将结束。

在调试透视图中有一些功能按钮，用来实行单步程序执行，见图 3-59 所示。

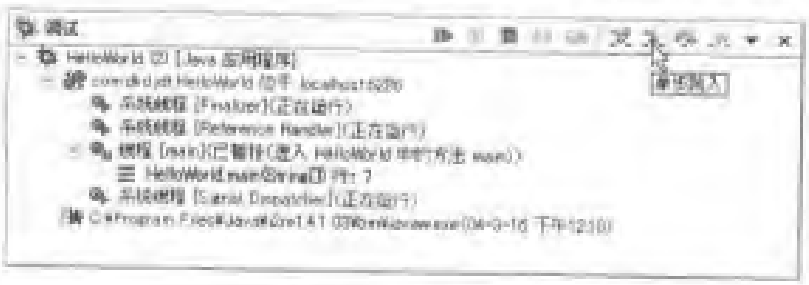


图 3-59　"调试"视图

下面是几种单步执行程序的方式：

（1）单步跳过。在“调试”视图中选择堆栈帧。该堆栈帧中执行的当前行在“调试”透视图的编辑器中突出显示。单击“调试”视图工具栏中的“单步跳过”按钮，或者按F6键。执行当前所选的行，并会在下一个可执行上暂挂。

（2）单步跳入。在“调试”视图中选择堆栈帧。所选帧中执行的当前行在“调试”透视图的编辑器中突出显示。单击“调试”视图工具栏中的“单步跳入”按钮，或者按F5键，调用在要执行的当前所选行上的下一个表达式，而执行会在调用的方法中的下一个可执行上暂挂。

（3）单步跳入选择。在“调试”视图中选择堆栈帧。所选帧中执行的当前行在“调试”透视图的编辑器中突出显示。在Java编辑器中，在执行的当前行中，将光标放置在想要单步跳入的方法的名称上。单击“运行”菜单或Java编辑器上下文菜单中的“单步跳入选择”操作，或者按“Ctrl+F5”键，将继续执行，直到调用了所选择的方法为止。

（4）使用过滤器单步执行。在“调试”视图中选择堆栈帧。所选帧中执行的当前行在“调试”透视图的编辑器中突出显示。单击“调试”视图工具栏中的“使用过滤器单步执行”按钮，或者使用“Shift+F5”快捷键，调用当前选择的要执行的下一个表达式，执行会在未过滤掉的下一个已执行的方法中的下一个可执行行上暂挂。设置单步过滤的方法：打开“窗口”→“首选项”→“Java”→“调试”→“单步执行过滤器”。

（5）运行至返回。在“调试”视图中选择堆栈帧。所选帧中执行的当前行在“调试”透视图的编辑器中突出显示。单击“调试”视图工具栏中的“运行至返回”按钮，或者按“F7”键，将继续执行，直到执行当前方法中的下一个返回语句，并且执行会在下一个可执行上暂挂。

（6）运行至行。暂挂线程时，可以继续执行，直到指定的行为止。这是在某行暂挂执行而不用设置断点的简便方法。将光标放在想要程序运行的行上，在弹出菜单中选择“运行至行”，或者使用“Ctrl+R”快捷键。程序将继续执行，在要执行的指定行之前暂挂。断点和异常可导致线程在到达指定行之前被暂挂。

5. 检查值

当选择了堆栈帧时，可以在“变量”视图的该堆栈帧中看到可视变量。“变量”视图显示基本类型的值，可通过展开复杂变量以显示它们的成员来检查复杂变量。

3.4.5 远程调试

Java调试器的客户机/服务器设计允许用户对网络中的计算机启动Java程序，并从运行平台的工作站调试该程序。远程调试对于在为不能主管开发平台的设备开发程序时尤其有用，当在专用机器（如Web服务器）上调试程序时，这一点也很有用。

注意：要使用远程调试，必须使用支持此功能的JVM。

要以远程方式调试程序，必须能够在远程机器上以调试方式启动程序，以便它将等

待来自调试器的连接。用于启动程序和连接调试器的特定技术是特定于虚拟机的。基本步骤如下：

- 确保正在以可用的调试信息构建 Java 程序。可以通过“窗口”→“首选项”→“Java”→“编译器”控制这些属性。构建 Java 程序后，将其安装到目标计算机。这涉及将.CLASS 文件或.JAR 文件复制到远程计算机上的适当的位置。
- 在远程计算机上调用 Java 程序，并使用适当的 VM 自变量来指定调试方式和调试器的通信端口。
- 使用远程启动配置来启动调试器并指定远程计算机的地址和端口。
- 应该从 VM 供应商那里获取用来设置进行远程调试启动配置的更特定信息。

远程 Java 应用程序启动配置应在正在远程 VM 上运行的应用程序时使用。因为应用程序是在远程系统上启动的，所以启动配置不会指定有关 JRE、程序自变量或 VM 自变量的普通信息，不过提供了有关连接至应用程序的信息。要创建远程 Java 应用程序启动配置，可执行下列操作：

（1）从工作台菜单栏中选择“运行（R）”→“调试...”，或者从工具栏中调试按钮的下拉菜单中选择“调试...”来显示“启动配置”对话框。

（2）在左边的配置类型列表中选择“远程 Java 应用程序”。

（3）单击“新建”按钮，就创建了新的远程启动配置，见图 3-60，画面上显示三个选项卡：连接、源代码和公共。

（4）在“连接”选项卡的“项目（P）”字段中，输入或通过“浏览（B）...”来选择用作启动的参考项目（用于源查找）。在“主机”字段中，输入 Java 程序正在其中运行的主机的 IP 地址或域名。如果程序与工作台在同一台机器上运行，则输入 localhost。在“端口”字段中，输入远程 VM 正在接受连接的端口。通常，启动远程 VM 时就要指定此端口。“允许终止远程 VM（A）”标志是一个切接开关，它确定是否在调试器中启用终止命令。如果想要终止正连接的远程 VM，则选择此选项。

（5）全部设置完后，单击“调试”（U）。启动将尝试连接至位于指定地址和端口的 VM，其结果显示在“调试”视图中。如果启动器无法连接至位于指定地址的 VM，则会出现错误消息。

（6）与 VM 断开连接，单击视图的工具栏中的“断开连接”按钮，与 VM 的通信终止，而远程 VM 中的所有线程仍继续进行。尽管远程 VM 继续执行，但调试会话已经终止了。

3.4.6　对表达式和代码段求值

JDT 添加了代码片段编辑测试窗设施，可以用来在构建完整 Java 程序之前对 Java 代码片段进行实验和求值。在“代码片段编辑测试窗”页面编辑器中对代码片段进行编辑和求值，检查和显示 Java 表达式，产生的问题在编辑器中报告。

从 Java 代码片段编辑测试窗编辑器中，可以选择代码片段、对其进行求值并以字符串形式显示结果。还可以在调试器的表达式视图中显示对代码片段求值所产生的对象。



图 3-60 远程启动配置

1. 创建 Java 代码片段编辑测试窗页面

代码片段编辑测试窗允许在调试器的控制下运行、检查和显示 Java 表达式。断点和异常的行为将像在常规调试会话中一样。在代码片段编辑测试窗页面打开之后，首次在其中对表达式求值时，会启动 VM。页面的 VM 将保持活动状态，直到（在调试器中或者通过编辑器工具栏的“停止求值”按钮）显式关闭、终止该页面为止，或者在对 System.exit()进行了求值时，VM 会终止活动状态。

创建 Java 代码片段编辑测试窗页面可以通过使用下面的方式来打开“新建 Java 代码片段编辑测试窗页面”向导。

（1）从“Java 透视图”的菜单栏中，选择“文件”→“新建”→“代码片段编辑测试窗页面”。如果是从其他非“Java 透视图”的菜单栏中，选择“文件”→“新建”→“其它”。然后在左边选择“Java”，在右边选择“代码片段编辑测试窗页面”。

（2）单击“下一步”。一旦打开了“新建 Java 代码片段编辑测试窗页面”向导，如图 3-61 所示。在“输入或选择文件夹（E）”字段中，输入或通过单击“浏览（W）”来选择新页面的容器。本例中的缺省文件夹为“JDT-ch3/com/dkd/jdt/test”。

（3）在“文件名”字段中，输入新页面的名称。如果自己不想输入扩展名，则系统将自动为其添加该扩展名.jpage。

（4）完成上述步骤后单击“完成”，这时新的代码片段编辑测试窗页面在编辑器中打开。



图 3-61　新建 Java 代码片段编辑测试窗页面

2. Java 代码片段编辑测试窗页面

代码片段编辑测试窗允许在调试器的控制下运行、检查和显示 Java 表达式。“Java 代码片段编辑测试窗页面”按钮功能如表 3-1 所示：

表 3-1　Java 代码片段编辑测试窗页面按钮功能表

命令	名　称	描　述
	运行代码片段	运行表达式对象对表达式求值，但不会显示结果
	显示	将对表达式求值的结果作为字符串显示在代码片段编辑测试窗编辑器中
	检查	将显示对“表达式”视图中的表达式进行求值的结果
	终止	此命令终止用来对表达式求值的 Java VM
	设置“导入声明”	此命令设置用于评估代码的上下文的导入声明

3. 检查对表达式求值的结果

检查并显示对“表达式”视图中表达式进行求值的结果。在代码片段编辑测试窗页面中，输入表达式或突出显示要检查的现有表达式。例如，“System.getProperties();”，单击工具栏中的“检查”按钮（或者从选择的弹出菜单中选择“检查”），检查结果出现在“表达式”视图中，如果还看不到它，则打开它。可以像检查调试器中的变量那样来检查结果。如图 3-62 所示，在对代码片段编辑测试窗页面中的表达式进行“检查”后，在“表达式”视图中显示结果。

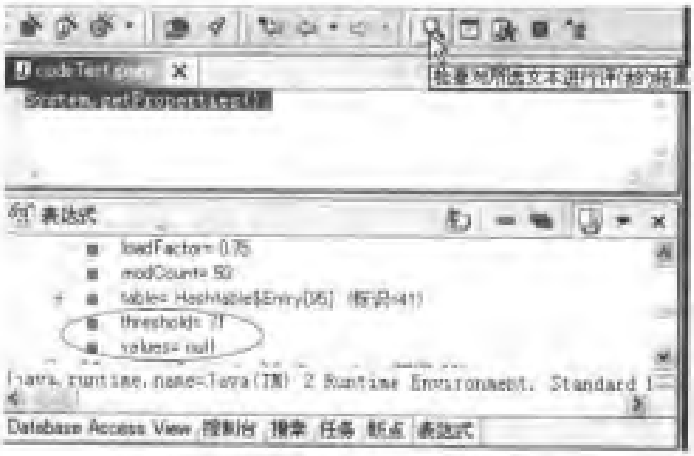


图 3-62　检查对表达式求值的结果

4. 显示对表达式求值的结果

按“显示”按钮将对表达式求值的结果显示在代码片段编辑测试窗编辑器中。在代码片段编辑测试窗页面中，输入表达式或者突出显示要显示的现有表达式。例如，“System.getProperties();”，单击工具栏中的“显示”按钮（或者从选择的弹出菜单中选择“显示”），求值结果会在代码片段编辑测试窗编辑器中突出显示。显示的结果可以通过将 toString()发送至求值的结果而获得的值，或者是对基本数据类型（比如，int）求值时，其结果是返回结果中的值。

例如，在编辑器中输入 new java.util.Date().toString()后，并单击“显示”按钮。则显示在代码片段编辑测试窗编辑器中的结果是：“（java.lang.String）Mon Apr 19 15:50:35 CST 2004”，如图 3-63 所示。



图 3-63　显示在代码片段编辑测试窗编辑器中的结果

或者是对基本数据类型（比如，int）求值时，例如，在编辑器中输入并突出显示（int）5*5，并按“显示”按钮。结果“(int) 25”将突出显示在该编辑器中，如图 3-64 所示。

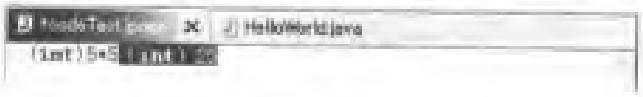


图 3-64　对基本数据类型求值

5. 运行代码片段

执行表达式时会对表达式求值，但是不会显示结果。如果选择要执行的表达式，并单击工具栏中的“执行”按钮，则不会显示任何结果，但是会执行代码。例如，如果你输入并突出显示 System.out.println("Hello World")，并单击“执行”按钮，则 Hello World 将出现在“控制台”视图中，但是没有任何结果显示在代码片段编辑测试窗编辑器或“表

达式”视图中，如图 3-65 所示。

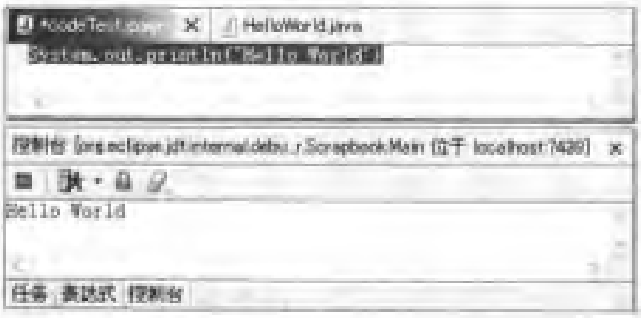


图 3-65　运行代码片段

6. 使用代码辅助

与常规 Java 编辑器相似，代码片段编辑测试窗编辑器也支持代码辅助。例如，在代码片段编辑测试窗编辑器中输入“System.out.”时会出现代码辅助列表，代码辅助将提示你可能的补全项，如图 3-66 所示。

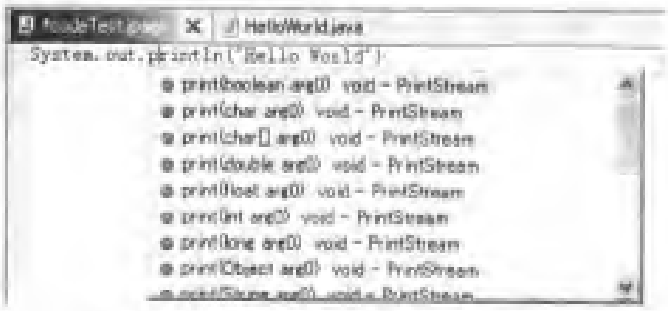


图 3-66　代码片段编辑测试窗使用代码辅助

7. 查看编译错误

如果对包含编译错误的表达式进行求值，则将在代码片段编辑测试窗编辑器中报告编译错误。例如，在编辑器中输入并选择表达式 System.println("hello")，并单击工具栏中的“执行”按钮。这时，错误消息“没有为类型 System 定义方法 println(String)”会突出显示在编辑器中出现错误的位置，如图 3-67 所示。

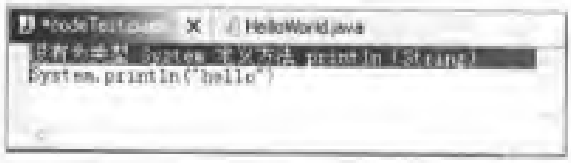


图 3-67　查看编译错误

8. 查看运行时异常

如果求值的表达式导致运行时异常，则在编辑器中将报告该异常。例如，在编辑器中输入并选择表达式“Object x = null; x.toString()”，并单击工具栏中的“显示”按钮。

则以下错误消息："在求值时发生异常：java.lang.NullPointerException"将在编辑器中显示。

3.5 JDT 搜索

Java 搜索允许用户查找 Java 元素（包、类型、方法和字段）的声明、引用和出现。搜索是通过后台提供的最新的索引来支持的，当与 Java 元素相应的资源发生更改时，后台仍然能保持最新的索引。对工作区执行的 Java 搜索与它们的构建状态无关。例如，可以在关闭了自动构建的情况下执行搜索。

3.5.1 从工作台执行 Java 搜索

1. 打开"搜索"对话框

在"搜索"对话框中，可以执行文件、文本或 Java 搜索。Java 搜索是对代码结构进行的。按名称或文本内容来对文件执行文件搜索。Java 搜索的速度更快，这是因为代码结构具有底层索引结构。文本搜索允许用户在注释和字符串内查找匹配，因此可以快速查找对 Java 元素的引用和声明。在"Java"透视图中，可通过下列任何一种方法来打开"搜索"对话框。

- 单击工具栏中的"搜索"按钮；
- 按"Ctrl+H"快捷键；
- 从菜单栏中选择"搜索（A）"→"搜索（A）..."。

这时会打开如图 3-68 所示的"搜索"对话框。

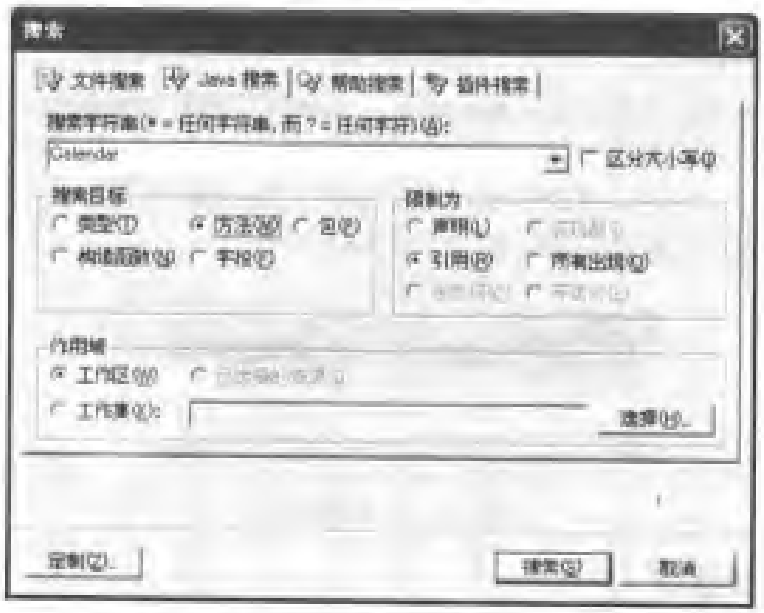


图 3-68 "搜索"对话框

2. 执行 Java 搜索

在“搜索”对话框中，选择“Java 搜索”选项卡，在“搜索字符串”字段中，输入用户搜索的字段，例如：Calendar，在“搜索目标”区域中，选择“方法（M）”，并在“限制为”区域中，选择“引用（R）”，在“作用域”中选择为“工作区（W）”，然后单击“搜索（S）”。

3. 执行文件搜索

在“搜索”对话框中选择“文件搜索”选项卡，如图 3-69 所示。在“包含文本（O）”字段中，输入用户想搜索的字段，例如“calendar”，在“文件名模式（P）”字段设置想要在哪种文件名模式中搜索，例如输入“*.java”和“*.js”（模式之间用逗号隔开），表示要在 Java 文件和 js 文件中搜索，“作用域”设置为“工作区（W）”，然后单击“搜索（S）”按钮。

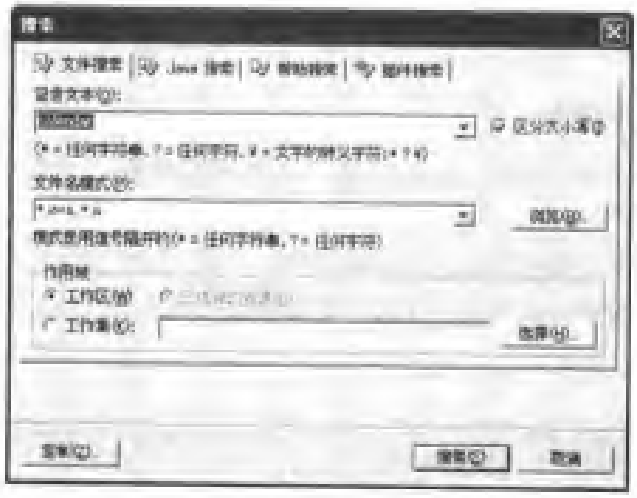


图 3-69　执行文件搜索

4. 显示搜索结果

在搜索过程中，随时可以单击“取消”来停止搜索，这时将会显示部分结果。在“Java”透视图中的“搜索”视图中显示搜索结果。使用“显示下一个匹配”和“显示上一个匹配按钮”来浏览每个搜索结果。当选中“搜索”视图中的搜索结果时，双击该搜索结果，将在编辑器中打开该搜索结果所属的文件，并停留在该搜索结果位置上。垂直标尺中的搜索标记对搜索结果作了标记，如图 3-70 所示。

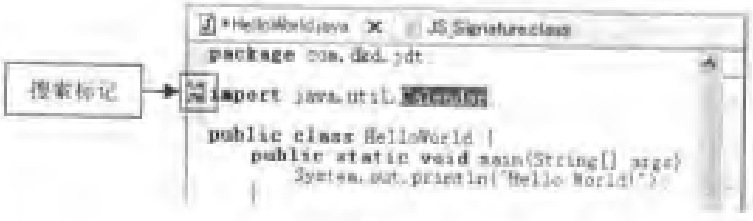


图 3-70　搜索标记

5. 查看先前的搜索结果

在“搜索结果”视图中，单击“上次搜索结果”工具栏按钮旁边的箭头来查看包含最近搜索列表的菜单，如图 3-71 所示，可以从此菜单中选择并查看先前的搜索结果。可以通过选择“清除历史记录（C）”来清除此列表。

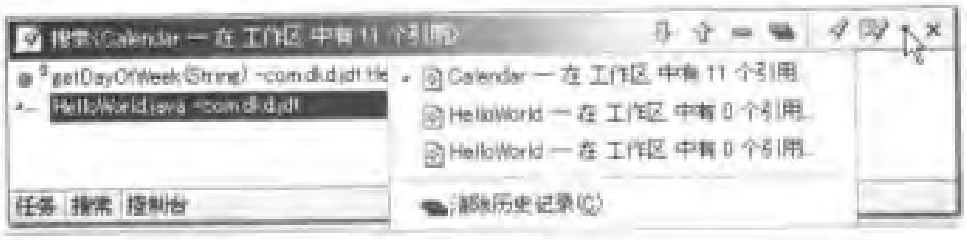


图 3-71 “搜索结果”视图

3.5.2 使用弹出菜单执行 Java 搜索

1. 在 Java 视图中搜索

用户可以在特定视图中执行 Java 搜索，这些视图包括“大纲”视图、“层次结构”视图、“包资源管理器”视图、“搜索结果”视图及“浏览”视图。在这些视图中，通过所选资源和元素的上下文菜单来执行 Java 搜索。如图 3-72 所示，在“包资源管理器”视图中，选择 getDayOfWeek(String)方法，单击鼠标右键，从它的上下文菜单中选择“声明（L）”→“工作区（W）”。



图 3-72 在 Java 视图中搜索

2. 在编辑器中搜索

在编辑器中，也可以通过所选 Java 元素的上下文菜单来进行 Java 搜索，如图 3-73 所示，选择类名 HelloWorld，右击鼠标，从它的上下文菜单中选择“搜索（S）”→“声明（L）”→“工作区（W）”。



图 3-73　从编辑器中搜索

3.6　重构支持

JDT 提供了重构工具，可以帮助用户很方便地重构代码。Java 程序重构的目标是在不影响程序行为的情况下进行系统范围内的代码更改。许多 Java 视图（例如，“包资源管理器”和“大纲”）和编辑器的上下文菜单都提供了重构命令。许多简单的命令，如“移动”和“重命名”实际上都是重构操作，这是因为移动和重命名 Java 元素通常需要更改从属文件。

当执行重构操作时，可以选择在执行重构操作之前预览由重构操作导致的所有更改。在预览重构操作时，将通知用户潜在的问题，并且向用户显示重构操作将执行的更改的列表。如果不预览重构操作，则将直接进行更改，并且将显示产生的任何问题。如果检测到不允许重构继续的问题，则操作将停止，并且将显示问题列表。

下面介绍 JDT 几种常用的重构方法。

3.6.1　Java 元素重命名

进行 Java 元素重命名时，重构操作更改代码的结构而不会更改其语义行为。重命名

方法如下：

（1）在“包资源管理器”视图中，选择 HelloWorld.java。

（2）从它的上下文菜单中，选择“重构（T）”→“重命名”。

（3）在“重命名编译单元”对话框中的“输入新名称（E）”字段中，输入“Hello World2”，如图 3-74 所示。

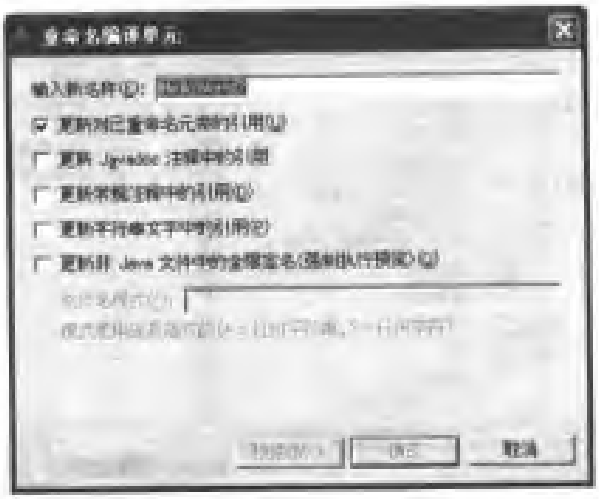


图 3-74 “重命名编译单元”对话框

（4）预览重命名类导致的更改。要预览重命名类导致的更改，按“预览（W）”。工作台将分析建议的更改，并提供重命名此资源时将进行的更改的预览。因为重命名编译单元将影响其他编译单元中的导入语句，所以更改会影响到其他编译单元。这些将显示在预览窗格中的更改列表中，如图 3-75 所示。在“重构”预览页面上，可以滚动浏览建议的更改，并在必要时选择或取消选择更改。通常情况下是将接受建议的所有更改。

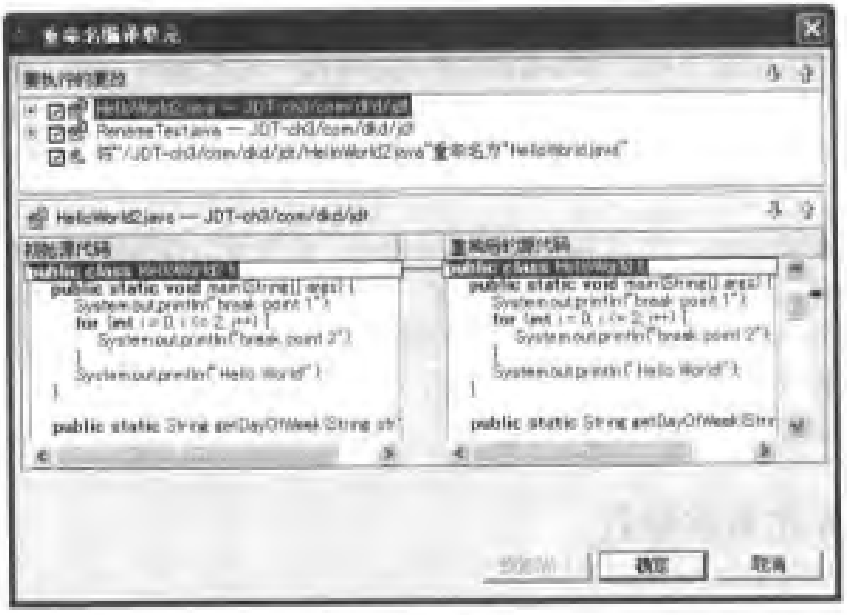


图 3-75 “重构”预览页面

（5）确定更改。单击“确定”以接受建议的所有更改。重构操作可能会导致不同编译单元发生许多更改，例如，为了说明重命名所执行的重构操作，现在又做了一个简单的例子 RenameTest.java。在该类中调用类中的 getDayOfWeHelloWorld.javaek()方法，当 HelloWorld.java 重命名后，将会看到 RenameTest.java 中跟 HelloWorld 有关的内容也发生了相应的改变，如图 3-76 所示。

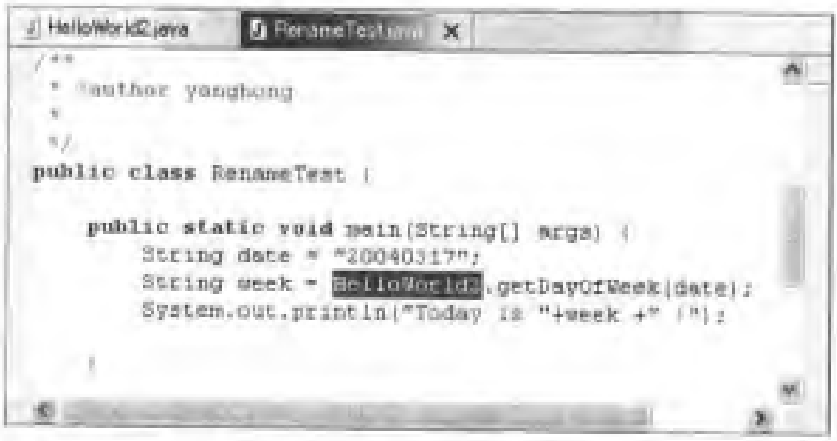


图 3-76　重构操作导致不同编译单元发生更改

（6）更改撤销。用户可以成组地撤消上述的更改。在菜单栏中，选择“重构（T）”→“撤销—将 HelloWorld.java 重命名为 HelloWorld2.java”，如图 3-77 所示。撤销重构更改，工作台则返回到它先前的状态。



图 3-77　撤销重构更改

3.6.2　Java 元素移动

和 Java 元素重命名一样，Java 元素移动和复制也将使用重构在 Java 包之间移动资

源，重构操作更改代码的结构而不会更改其语义行为，具体操作如下。

（1）在“包资源管理器”视图中，选择 RenameTest.java 文件并将其拖动到 com.dkd.jdt.test 包中，或者从菜单栏或从上下文菜单中选择“重构（T）”→“移动”。

（2）系统提示用户选择是否要更新对正在移动的文件的引用，如图 3-78 所示。通常，用户将想要执行此操作以避免编译错误。

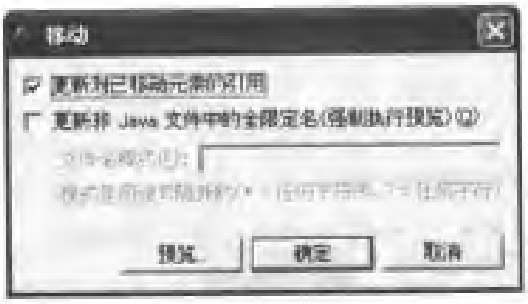


图 3-78 “移动”对话框

（3）按“预览”按钮来查看移动操作将导致的更改的列表，如图 3-79 所示。

（4）按预览窗口中的“完成”或在提示对话框按“确定”按钮，该文件就移到了指定的 com.dkd.jdt.test 包里，并且其包声明更改为相应的新包名。另外，程序中调用原来同属本包的 HelloWorld.java 类中的方法，现在因为不在同一包内，而需要增加的声明“import com.dkd.jdt.HelloWorld;”也自动添加到新代码中。

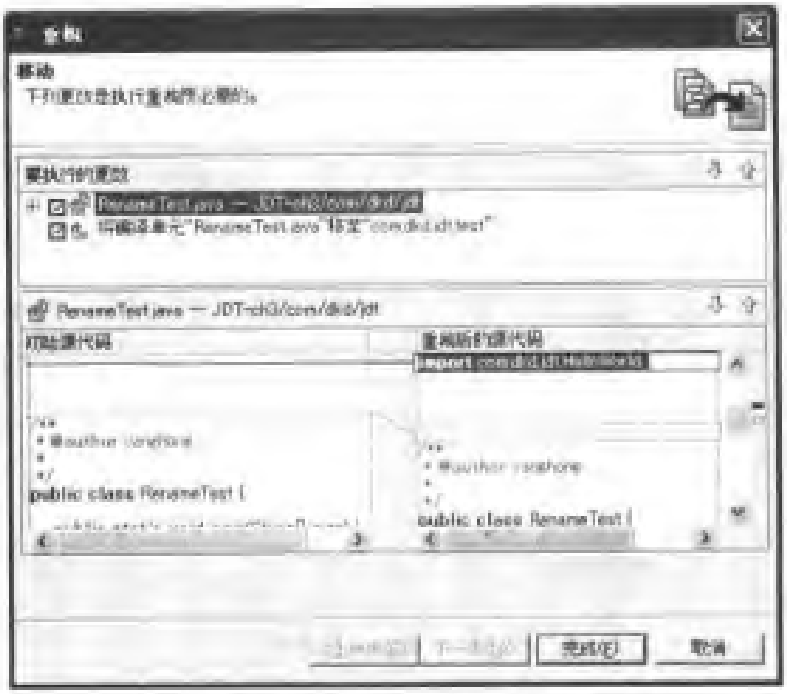


图 3-79 “重构”预览页面

如果将文件移至缺省包，系统将另外弹出一个警告对话框，将移至缺省包时所作的

引用更新通知用户。如图 3-80 所示。如果不想每次将文件移至缺省包时都看到该对话框，则选择“不再显示此消息”复选框。



图 3-80　移动警告对话框

上下文菜单是使用拖放操作的备用方法。使用菜单时，除了选择已看到的更新首选项以外，必须在“移动”对话框中指定目标包。在“移动”对话框中，展开层次结构，以浏览资源可能的新位置。选择所要移至的新位置，然后单击“确定”即可。

3.6.3　Java 元素复制

要复制 Java 元素，需要将它们复制至剪贴板，并在希望的新位置粘贴它们。操作步骤如下：

（1）从 Java 视图中，选择想要复制至剪贴板的 Java 元素，点鼠标右键，在上下文菜单中选择“复制”或按“Ctrl+C”快捷键。

（2）选择所要粘贴的位置，在上下文菜单中选择“粘贴”或者按“Ctrl+V”快捷键，则复制完毕。

还可以通过将 Java 元素拖放到期望的新位置来复制 Java 元素。在通过拖动来复制元素时，需要按住 Ctrl 键。

本节只简单介绍了几种常用的简单的重构操作，重构操作还包括很多内容，比如抽取方法、抽取常量、封装字段、内联、变量转换等，在本书的后面还将对 Eclipse 关于重构方面的内容进行详细的讨论。请读者参阅第 13 章“使用 Eclipse 进行重构”。

3.7　Java 构建器

Java 构建器使用“Java 语言规范”的编译器来构建 Java 程序。在保存 Java 文件时，Java 构建器能够以增量方式构建程序。

编译器检测到的问题分为警告或错误。警告不会影响程序的执行，出现错误（如“Java 语言规范”所指定的错误）时，Java 编译器将会在编译时报错。然而，对于一些其他类型的问题，用户可以根据需要来指定 Java 编译器是将它们作为警告或是错误来报告，还是忽略它们。

即使在存在编译错误的情况下，Java 编译器也可以创建 CLASS 文件。然而，在存

在严重错误的情况下（例如，引用与 Java 编译器不一致的二进制文件，会导致构建路径无效），Java 构建器不会生成任何 CLASS 文件。

3.7.1 构建 Java 程序

构建命令将编译工作台资源。可以通过自动构建或者手工构建来触发构建命令。

1. 自动构建

要启用自动构建，选中“窗口”→“首选项”→“对资源修改时自动执行构建（B）”复选框，如图 3-81 所示，这样每当用户保存已修改的工作台资源时，系统就会自动进行增量构建。推荐使用自动构建。

要禁用自动构建，则清除“窗口”→“首选项”→“对资源修改时自动执行构建（B）”复选框即可。

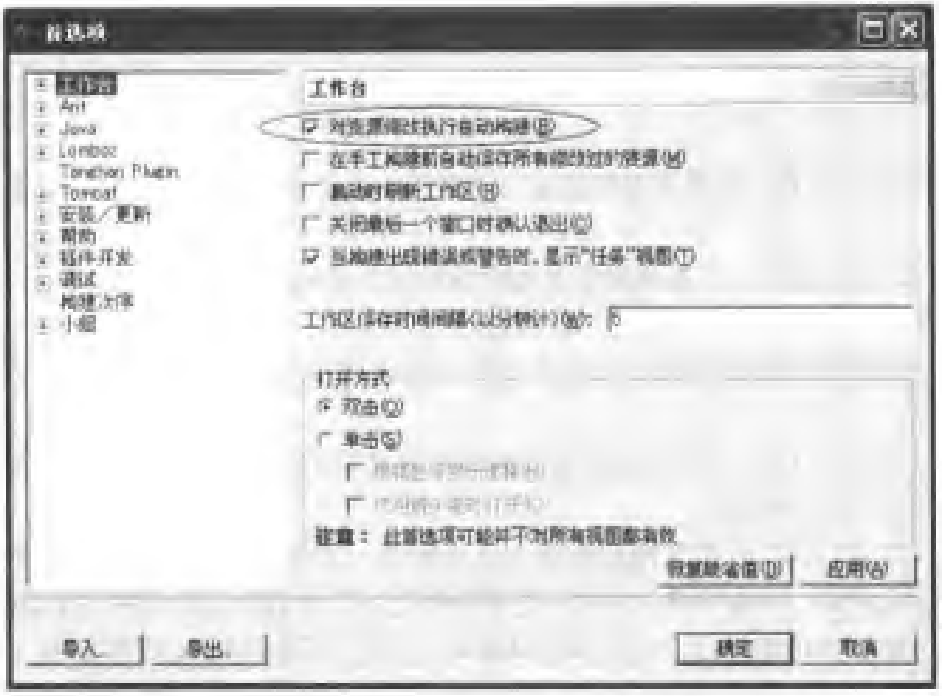


图 3-81 选择自动构建

2. 手工构建

可以使用键盘快捷键、项目的上下文菜单以及菜单栏中的“项目”菜单来执行手工构建。

（1）增量式构建：这种构建会编译自上次构建后修改过的所有资源。要实现增量构建，可从菜单栏中选择“项目”→“全部构建（L）”或者按“Ctrl+B”快捷键方式。

（2）增量式项目构建：可以增量式构建单个项目，选择想要构建的项目，从菜单栏中选择“项目”→“重建项目（R）”。

（3）完整构建：要重建具有相关联的构建器的所有工作台资源（例如，Java 项目），

同增量式构建一样，从菜单栏中选择“项目”→“全部重建（L）”。

（4）完整项目构建：要重建项目中包含的所有资源，选择该项目，然后从菜单栏中选择“项目”→“重建项目（R）”。

3. 构建循环项目

（1）启用构建循环项目：选择“窗口”→“首选项”→“Java”→“编译器”→“构建路径”页面，然后将“循环依赖项”选项设置为“警告”，如图 3-82 所示。

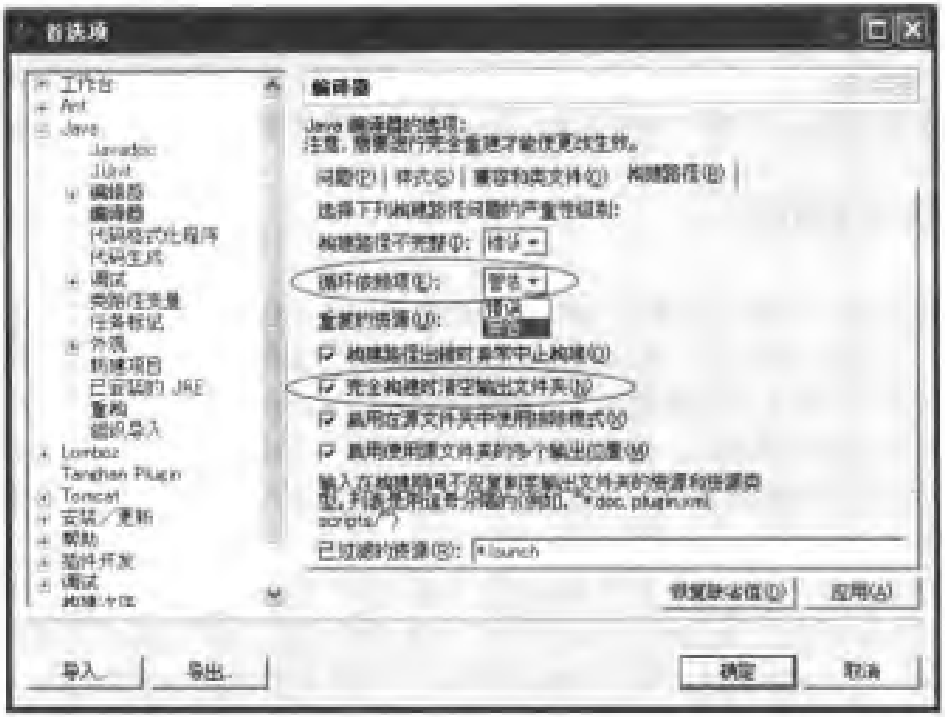


图 3-82　“构建路径”页面

若要禁用构建循环项目，则将“循环依赖性”选项设置为“错误”即可。

（2）启用构建涉及到循环的单个项目：选择项目，打开项目的“属性”对话框中，菜单栏的“项目（P）”→“属性（P）”→“Java 编译器”→“构建路径”，然后将“循环依赖项”选项设置为“警告”。

同样地，要禁用构建涉及到循环的单个项目，则将“循环依赖项”选项设置为“错误”即可。

4. 构建而不清除输出位置

要构建项目而不清除输出位置，选择“窗口”→“首选项”→“Java”→“编译器”→“构建路径”页面，如图 3-82 所示，然后选中“在完全构建时清空输出文件夹”复选框。

要构建单个项目而不清除输出位置，则选择该项目，打开项目的“属性”对话框中，菜单栏的“项目（P）”→“属性（P）”→“Java 编译器”→“构建路径”，然后选中“完全构建时清空输出文件夹”复选框。

5. 查看编译错误和警告

工作台支持使用不同方法来检查错误和警告，具体包括：

- Java 视图（例如，“包资源管理器”）中的错误记号
- 编辑器中的标记注释
- 任务列表中的任务

如果你选择对其求值的表达式存在编译错误，则将在“代码片段编辑测试窗”编辑器中报告该错误。

例如，在 3.4.6 节中所举的例子，如果在编辑器中输入并选择（无效）表达式 System.println("hello")，单击工具栏中的“运行”按钮时，则在编辑器中发生错误的地方会显示以下错误消息：“没有为类型 java.lang.System 定义方法 println(java.lang.String)。”

3.7.2 构建类路径

当进行 Java 开发时，设置正确的 Java 构建路径是一项重要任务。没有正确的 Java 构建路径，就不能编译代码，也不能搜索或查看 Java 元素的类型层次结构。构建类路径是用来查找源代码所引用的类的路径。在编译期间，使用此路径来搜索项目外部的类，为每个项目指定构建类路径。在项目属性中，将其称为“Java 构建路径”。Java 项目的构建路径可以包含源代码文件、其他 Java 项目和 JAR 文件。可以使用文件系统路径或通过使用引用网络位置的变量来指定 JAR 文件。

类路径变量允许用户避免引用本地文件系统上的 JAR 文件位置。通过使用类路径变量，可以只使用变量名（如 JRE_LIB）来指定 JAR 文件或库，而不是指定工作站上的 JRE 位置。这样，可以在小组之间共享构建路径和定义变量来引用特定计算机的正确位置。

1. 查看和编辑项目的 Java 构建路径

可以在使用“新建向导”来创建项目时定义项目的 Java 构建路径，也可以稍后在项目的属性对话框中定义。在这两种情况下，Java 构建路径设置对话框是相同的。要查看和编辑项目的 Java 构建路径，可遵循下列步骤：

（1）选择想要查看或编辑的项目。

（2）从项目的弹出菜单中选择“属性（P）”。

（3）打开“Java 构建路径”页面，包括四个选项卡，如图 3-83 所示。

（4）选择“源（S）”选项卡，在其页面上定义构建路径的源条目，可进行的操作有：

- “添加文件夹（A）”按钮可用来将源文件夹添加至 Java 构建路径；
- “除去（R）”按钮用来将从构建路径中除去所选择的文件夹；
- “编辑（E）”按钮允许用户修改选择的条目。

（5）选择“项目（P）”选项卡，在其页面上标识构建此项目所需要的其他项目。列表显示了工作台中的所有现有 Java 项目。注意，所选择的每个项目都会自动添加至所引用的项目的列表。

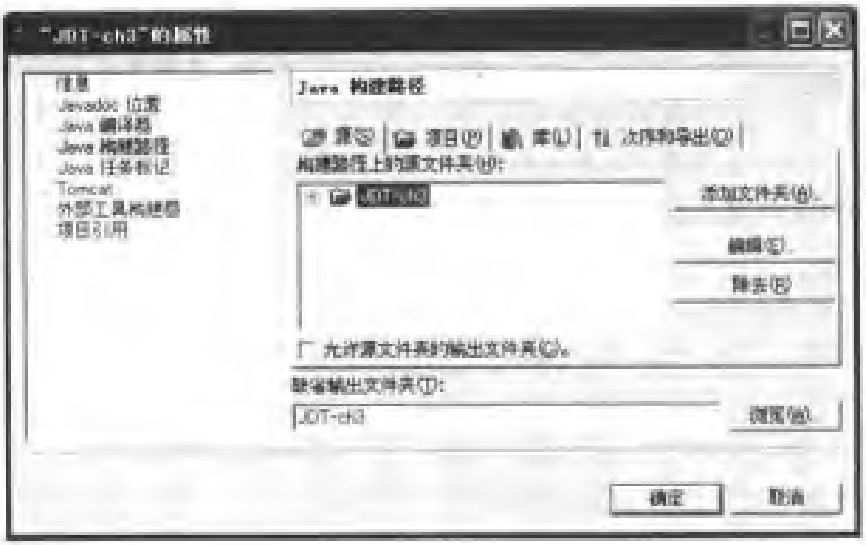


图 3-83　项目的 Java 构建路径

（6）选择“库（L）”选项卡，在其页面上定义项目所需要的库。库以不同的格式显示，有一些按钮用于添加每种格式的库。缺省情况下，每个 Java 项目在构建路径上都具有一个“JRE 系统库”条目，此条目代表工作台的缺省 JRE，如图 3-84 所示。



图 3-84　“JRE 系统库”条目

（7）选择“次序和导出（O）”选项卡，在其页面上定义 Java 构建路径次序。建议的顺序是将源条目放在库条目和所需的项目前面。

2. 将库文件夹添加至构建路径

库文件夹是一个常规文件夹，它包含工作台内的类文件的集合。在未将库封装为 JAR 文件时，对库就是使用这种格式。要将库文件夹添加到项目的构建类路径中，可遵循下列步骤。

（1）选择项目，在“Java 构建路径”页面中选择“库（L）”选项卡，见图 3-84。

（2）单击“添加类文件夹“按钮，在出现的对话框中，选择要添加的文件夹，并按“确定”按钮。如果是想要添加尚未存在的文件夹，首先使用“创建新文件夹”按钮，在创建了文件夹之后，选择该文件文件夹，并按“确定”按钮。

3. 将 JAR 文件添加至构建路径

可以将存储在工作台或者文件系统中的任何位置的 JAR 文件添加到构建类路径中。要将 JAR 添加到构建类路径中，遵循下列步骤：

（1）选择项目，在“Java 构建路径”页面中选择“库（L）”选项卡，见图 3-84。

（2）如果要添加工作区内的一个 JAR 文件，单击“添加 JAR”按钮，如果要添加外部 JAR 文件，单击“添加外部 JAR”按钮。

（3）在出现的对话框中，选择想要添加的 JAR 文件，一次可以添加多个 JAR。

4. 将类路径变量添加至构建路径

要将类路径变量添加至项目的 Java 构建路径，遵循下列步骤：

（1）选择项目，在“Java 构建路径”页面中选择“库（L）”选项卡，见图 3-84。

（2）单击“添加变量（V）”来添加涉及到 JAR 文件的变量。

（3）出现如图 3-85 所示的“新的变量类路径条目”对话框，它显示出所有可用的类路径变量。

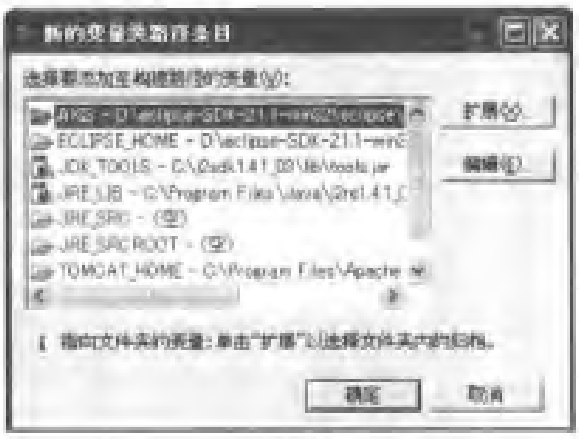


图 3-85　添加类路径变量

（4）选择类路径变量，然后按“确定”。

如果变量解析为一个文件夹，按“扩展（X）...”按钮，则可以指定指向 JAR 的路径扩展。按“编辑（E）...”按钮来创建新的类路径变量，或者编辑现有类路径变量。

5. 将源连接至 JAR 文件

可以将源连接至 JAR 文件，来对二进制 JAR 文件中包含的类启用源级单步执行和浏览。如果源代码没有连接至工作台中的 JAR 文件上，就无法查看 JAR 文件的源。要将源连接至 JAR 文件，遵循下列步骤：

（1）选择项目，在“Java 构建路径”页面中选择“库（L）”选项卡，见图 3-84。

（2）选择想要将源连接至的JAR文件。通过单击左边的加号来展开节点，并选择“源连接”节点。

（3）单击“编辑”按钮以显示源连接对话框，如图3-86所示。



图3-86　“源连接配置”对话框

（4）根据位置来填写“位置路径（C）”字段，在工作区、外部文件或外部文件夹之间进行选择。

5. 将源连接至库文件夹

可以将源连接至库文件夹，来对库文件夹中包含的类启用源级单步执行和浏览。如果源代码没有连接至工作台中的库文件夹，就无法查看库文件夹的源。要将源连接至库文件夹，遵循下列步骤：

（1）选择项目，在“Java构建路径”页面中选择“库（L）”选项卡，见图3-84。

（2）选择想要将源连接至的库文件夹。通过单击左边的加号来展开节点，并选择“源连接”节点。

（3）单击“编辑”按钮以显示源连接对话框。

（4）根据位置来填写“位置路径（C）”字段，在工作区、外部文件或外部文件夹之间进行选择。

6. 将源连接至类路径变量

连接类路径变量条目的源时，指向“归档”的路径和“根路径”都必须由变量定义。遵循下列步骤来对变量进行源连接：

（1）选择项目，在“Java构建路径”页面中选择“库（L）”选项卡，见图3-84。

（2）选择想要将源连接至的类路径变量。通过单击左边的加号来展开节点，并选择“源连接”节点。

（3）单击“编辑”按钮以显示源连接对话框。

（4）在“位置变量路径”字段中，使用变量来定义归档的位置。

（5）使用“变量”按钮来选择现有变量，使用“扩展”按钮来追加可选的路径扩展。

7. 定义类路径变量

类路径变量在工作台中是全局存储的。换句话说，工作台中的所有项目可以共享这些类路径变量。要添加或更改类路径变量，遵循下列步骤：

（1）从菜单栏中，选择“窗口”→“首选项”→“Java”→“类路径变量”，如

图 3-87 所示。在该页中允许用户添加、编辑或除去类路径变量。

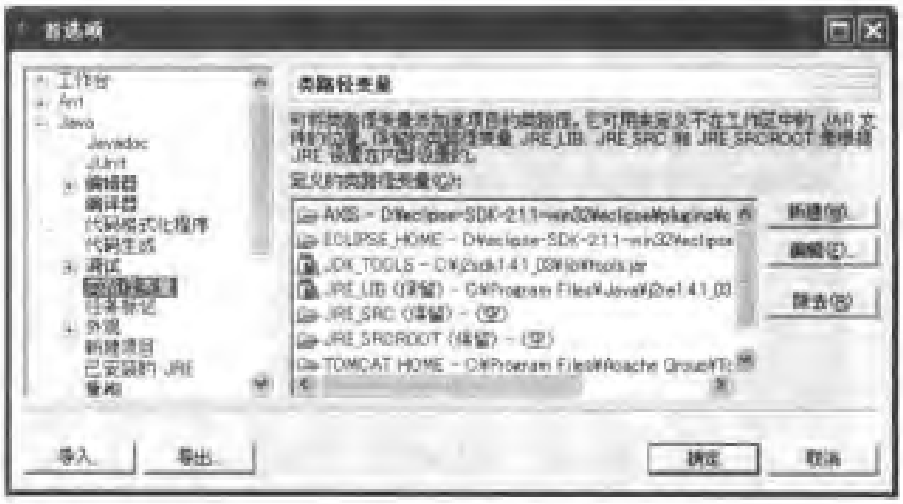


图 3-87 定义“类路径变量”窗口

（2）要添加新的类路径变量，单击“新建（W）...”按钮，打开“新建变量条目”页面。

（3）要编辑现有类路径变量，在“定义的类路径变量（C）”列表中选择该变量，并单击“编辑（E）...”按钮，打开“编辑变量条目”页面。

注意：在此页中不能编辑保留的类路径变量 JRE_LIB（保留）、JRE_SRC（保留）和 JRE_SRCROOT（保留）。要更改它们，应该在“已安装的 JRE”页面（“窗口”→“首选项”→“Java”→“已安装的 JRE”）上更改缺省工作台 JRE。

在如图 3-88 所示的“编辑变量条目”页面中，在“名称（N）”字段中输入变量的名称，在“路径（P）”字段中输入变量要引用的路径。还可以单击“文件（F）”或“文件夹（O）”按钮来浏览文件系统。完成时单击“确定”按钮，这时新的或已修改的变量便出现在“首选项”页面上的“已定义的类路径变量”列表中。

（4）要删除类路径变量，在“定义的类路径变量（C）”列表中选择该变量，并单击“除去（R）...”按钮。于是从首选项页面上的类路径变量列表中除去了该变量。

注意：不能删除保留的类路径变量 JRE_LIB、JRE_SRC 和 JRE_SRCROOT。



图 3-88 “编辑变量条目”页面

3.7.3　使用JRE

JRE的定义包含：

- JRE的类型（例如，“标准VM”或“标准1.x.x VM”）；
- 名称；
- 安装JRE的位置；
- Javadoc的位置；
- 包含Java系统类（如java.lang.Object）的系统库，及其他可选的系统库。这些系统库可以与源文件（它包含JRE的CLASS文件中的类的源）相关联。

缺省JRE是将预定义的类路径变量JRE_LIB，JRE_SRC和JRE_SRCROOT绑定的JRE。用户可以根据需要安装多个JRE，也可以切换工作台的缺省JRE。

1．指定工作台的缺省JRE

在所有项目中都使用缺省JRE来编译和启动Java程序，除非用户特地覆盖了缺省JRE。如果已经从JRE_LIB变量的构建路径中除去了该变量，则不会对缺省JRE编译项目。如果为缺省JRE的项目设置了定制运行时JRE，则不会使用缺省JRE来启动程序。

若想更改缺省JRE，可参照下面描述的方式进行更改缺省JRE。

（1）从工作台的菜单栏中，选择“窗口”→“首选项”→“Java”→“已安装的JRE”，如图3-89所示。

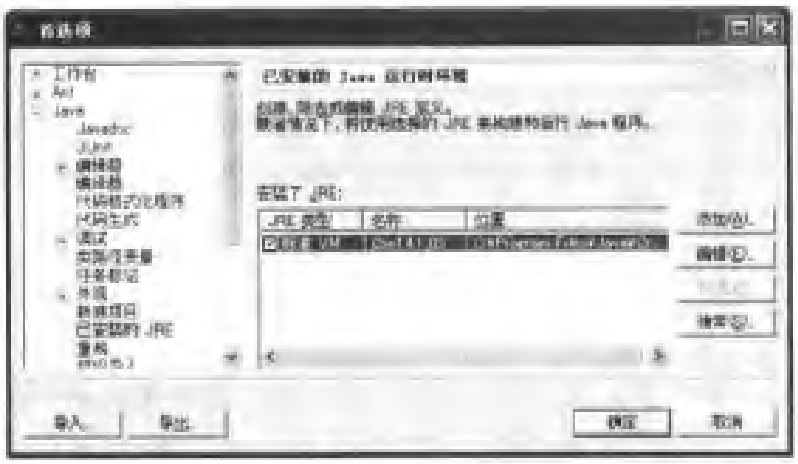


图3-89　指定工作台的缺省JRE

（2）选择想要指定为工作台中的缺省JRE的所对应的复选框。

（3）如果列表中未出现想要指定为缺省JRE的JRE，则必须添加它。

（4）完成时单击“确定”。

注意：当用户更改缺省JRE时，将自动更新系统变量JRE_LIB，JRE_SRC和JRE_SRCROOT。可以通过以下步骤来进行系统自动构建：“窗口”→“首选项”→“工作台”→“进行资源修改时自动执行构建”。

2. 添加新的 JRE 定义

添加新的 JRE 定义可遵循下列步骤：

（1）从工作台的菜单栏中，选择“窗口”→“首选项”→“Java”→“已安装的 JRE”，如图 3-89 所示。

（2）单击“添加（A）...”按钮打开“创建 JRE”对话框，如图 3-90 所示。

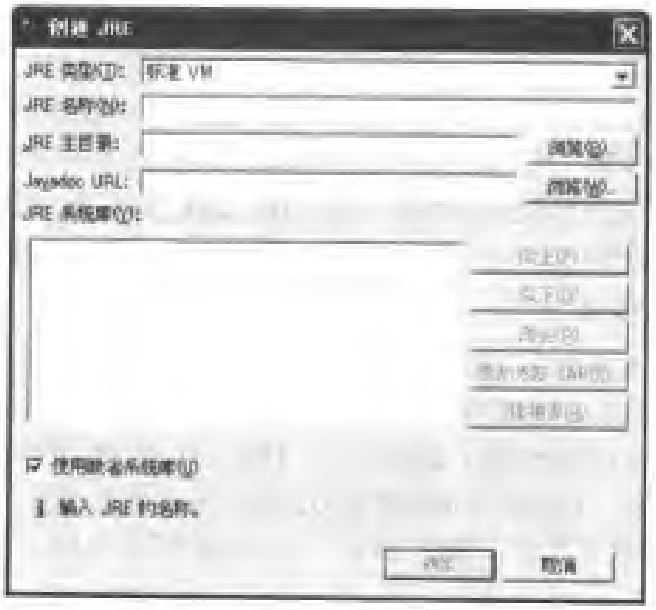


图 3-90 “创建 JRE”对话框

“创建 JRE”对话框各项说明如下：

- 在“JRE 类型（T）”字段中，从下拉列表中选择想要添加的 JRE 类型。
- 在“JRE 名称（N）”字段中，输入新的 JRE 定义的名称。相同类型的所有 JRE 都必须具有惟一名称。
- 在“JRE 主目录”字段中，输入或单击“浏览（B）...”按钮来选择指向 JRE 安装的根目录的路径（通常该目录包含 JRE 的 bin 和 lib 目录）。会自动选择此位置以确保它是有效路径。
- 在“Javadoc URL”字段中，输入或单击“浏览（W）”按钮来选择 URL 位置。Javadoc 导出向导将该位置用作缺省值，“打开外部 Javadoc”操作也使用该位置。

（3）如果想要对此 JRE 使用缺省库和源文件，则选择“使用缺省系统库（U）”复选框。否则，清除该复选框，并按需要进行定制。也可以对所引用的 JAR 连接源。

（4）完成时单击“确定”按钮。

3. 覆盖 JRE 定义的缺省系统库

添加 JRE 时，可以覆盖系统库以及相应的源连接，具体步骤如下。

（1）开始添加新的 JRE 定义。

（2）在“创建 JRE”对话框中，清除“使用缺省系统库（U）”复选框。如果系统能够确定系统库的设置，则将已经显示了这些库。

（3）对“JRE 系统库（Y）”里已确定的系统库，通过“向上（P）”或“向下（D）”按钮来进行排序，通过“除去（R）”除去系统库，通过“添加外部 JRE（X）”按钮将外部 JAR 添加到系统库的列表中。

（4）使用“连接源（H）”按钮来将源与系统库相关联。

（5）当完成此对话框时，单击“确定”以创建新的 JRE 定义。新的 JRE 定义将使用已定制的系统库（和已连接的源）。

4. 编辑 JRE 定义

除了 JRE 类型不可以修改以外，JRE 定义的所有设置都可以修改。编辑 JRE 定义可遵循下列步骤：

（1）从工作台的菜单栏中，选择“窗口”→“首选项”→“Java”→“已安装的 JRE”，如图 3-89 所示。

（2）选择想要编辑的 JRE 定义，并单击“编辑...”按钮打开“编辑 JRE”页面，如图 3-91 所示。

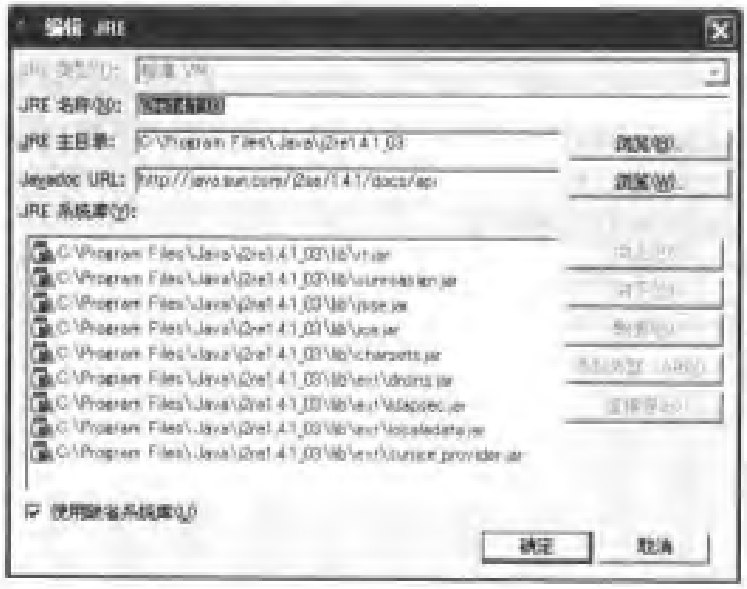


图 3-91　“编辑 JRE”对话框

图中各项说明如下：

- 在“JRE 名称（N）”字段中，编辑 JRE 定义的名称。相同类型的所有 JRE 都必须具有惟一名称。
- 在“JRE 主目录”字段中，编辑或者单击“浏览（B）...”按钮来选择指向 JRE 安装的根目录的路径（通常该目录包含 JRE 的 bin 和 lib 目录）。会自动选择此位置以确保它是有效路径。
- 在“Javadoc URL”字段中，编辑或者单击“浏览（W）...”按钮来选择 URL 位置。Javadoc 导出向导将该位置用作缺省值，“打开外部 Javadoc”操作也使用该位置。

（3）如果想要对此 JRE 使用缺省库和源文件，则选择“使用缺省系统库”复选框。否则，清除该复选框，并按需要进行定制。也可以对所引用的 jar 文件进行连接源操作。

（4）完成时单击“确定”按钮。

5. 选择用于启动配置的 JRE

用户可以专门指定用于启动个别配置的 JRE，而不必使用缺省 JRE 来运行和调试所有 Java 应用程序启动配置。选择用于启动配置的 JRE 可遵循下列步骤：

（1）在“启动配置”对话框中选择了 Java 应用程序配置之后，选择 JRE 选项卡。

（2）在可用的 JRE 的列表中，选择想要用来启动此配置的 JRE，并单击“应用”、“运行”或“调试”按钮。

注意：更改用于运行的 JRE 不会影响编译 Java 源的方式。可以对定制库调整要编译的构建路径。

本章小结

本章主要介绍了 Eclipse 中用于开发 Java 程序的 JDT 插件的功能及使用，包括：JDT 的基本操作、Java 编辑器的功能及使用、如何应用 JDT 的相关视图、程序的运行和调试、Java 搜索、重构支持及 Java 构建器的相关知识。相信通过本章的学习，读者完全可以在 Eclipse 上灵活方便地开发 Java 程序了。

在第 4 章中将介绍在 Eclipse 环境下用于开发 C 和 C++程序的插件——CDT 的有关内容，希望能对使用 C 和 C++语言编程的读者有所帮助。

第 4 章　利用 CDT 编写 C/C++程序

IDE 产生的重要意义在于协助程序员快速高效地完成编程工作。程序员通常不惧怕难以解决的问题，而是常常苦恼如何将一些重复性的工作变得简便，将大量简单的问题快速、正确地解决掉。本章中介绍的 CDT 插件就是用来解决程序员在 C/C++工程中的上述需求。

4.1　CDT 插件介绍

CDT 从字面含义上就可以看出，CDT 是开发 C/C++项目的工具箱。

安装 CDT 插件后，能够使 Eclipse 支持 C/C++项目的开发。Eclipse 主要支持 Java 及其相关项目的开发，但是，由于 Eclipse 本身的插件体系结构决定了它可以支持其他编程语言的开发。

CDT 提供了一套嵌入 C/C++统和开发环境的插件。它在 Eclipse 平台上增加了一个 C/C++透视图（Perspective），从而使 Eclipse 通过一些视图（View）、向导（Wizards）、编辑器（Editoir）、和除错工具（Debugger）来支持 C/C++开发。

CDT 是一个开放源代码项目（Open Source Projects），并通过公用公共许可（Common Public License）进行认证。

4.2　安装 MinGW

为了能够用 Eclipse 编译 C 和 C++程序，必须要安装一个 C/C++编译器，例如，Cygwin，MinGW。本书选用 MinGW。MinGW 是以 GNU 为基础的开发 C/C++项目的工具箱，能够提供 C/C++所需的头文件和库文件。

下载网址为 http://sourceforge.net/projects/mingw/。本书采用的版本是 MinGW-3.0.0-1.exe，请根据你的情况下载 MinGW。

下载 MinGW 的安装文件后即可按以下步骤安装 MinGW。

（1）首先双击安装文件，则出现如图 4-1 所示的对话框。



图 4-1　出现安装画面

（2）选择“是（Y）”后，出现安装向导对话框，如图 4-2 所示。



图 4-2　通过向导开始安装

（3）选择“Next”后出现使用条款对话框，如图 4-3 所示。

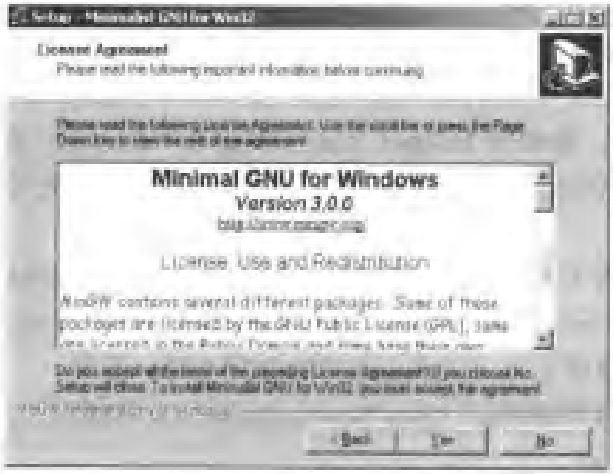


图 4-3　使用条款

（4）选择“Yes”后出现重要信息对话框，如图 4-4 所示。

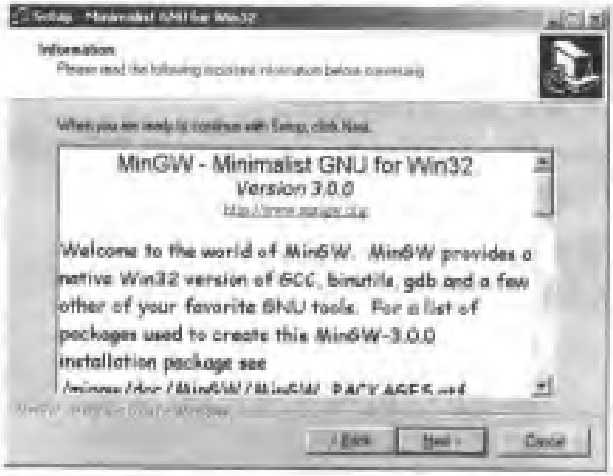


图 4-4　MinGW 的一些重要信息

（5）选择“Next”，出现如图 4-5 所示的对话框，该对话框是请用户选择安装 MinGW 的路径，本书选择的安装路径为：H:\MinGW。

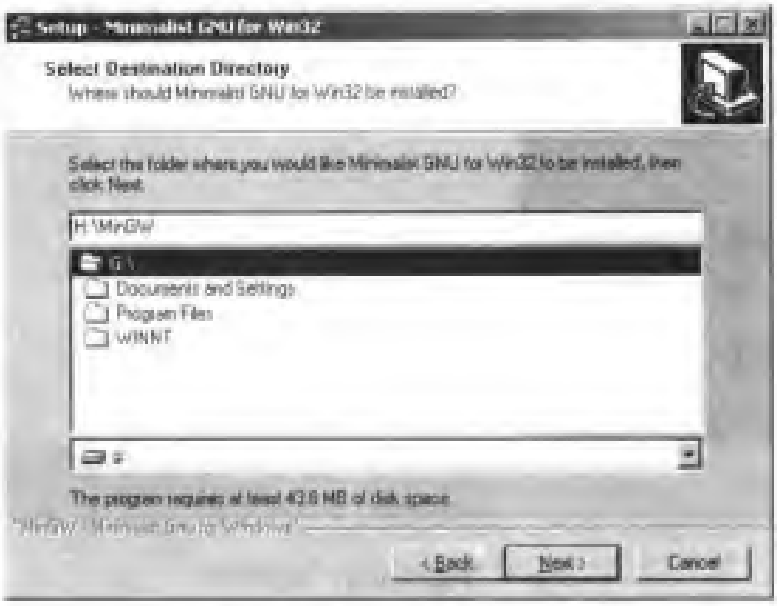


图 4-5　选择安装 MinGW 的路径

（6）选择“Next”，出现如图 4-6 的对话框，该对话框显示安装的路径。选择“Install”开始安装。



图 4-6　显示安装的路径

（7）安装完成后如图 4-7 所示，选择“Finish”。

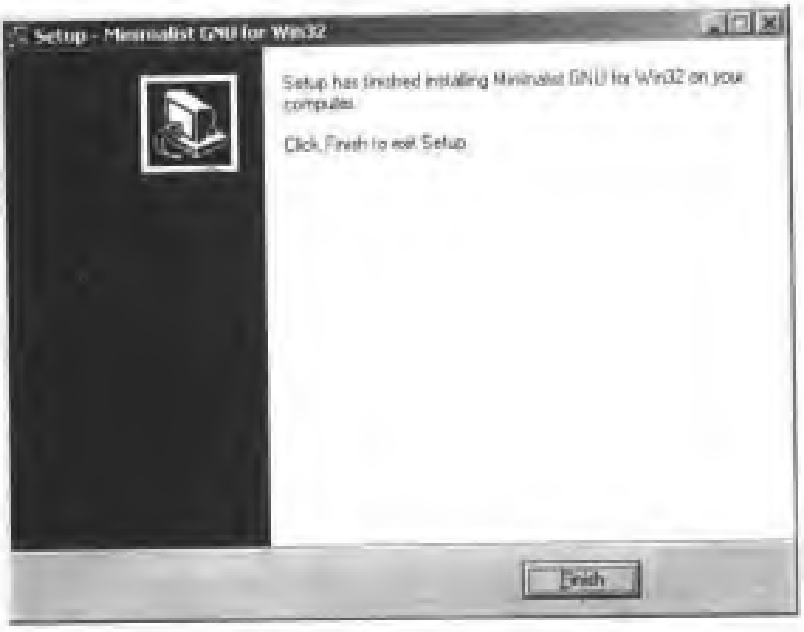


图 4-7　单击“Finish”安装结束

（8）为了能够不指明完整路径，系统就能够认识 MinGW 的执行命令，需要调整系统的环境变量。方法是：右击“我的电脑”，选择“属性（R）”→“高级”→“环境变量（E）”，选中“系统变量（S）”中的 Path 后选择“编辑（I）...”，在弹出的对话框中的“变量值”的最后填入“;H:\MinGW\bin”，如图 4-8 所示。之后单击“确认”。



图 4-8　编辑系统变量

（9）确认一下 MinGW 安装及环境变量设定是否生效。选择“开始”→“运行（R）...”，写入“cmd”→“确定”。输入“gcc -v”，如果屏幕出现如下几行信息表示安装及环境变量设定正确。

```
    Reading specs from H:/MinGW/bin/../lib/gcc-lib/mingw32/3.2.3/specs
    Configured with: ../gcc/configure --with-gcc --with-gnu-ld --with-gnu-as
--host=
    mingw32 --target=mingw32 --prefix=/mingw --enable-threads --disable-nls
--enable
    -languages=c++,f77,objc --disable-win32-registry --disable-shared
--enable-sjlj-
    exceptions
    Thread model: win32
    gcc version 3.2.3 (mingw special 20030504-1)
```

若屏幕出现如下一行信息，表示环境变量设定有问题，需要核对环境变量的设置。

'gcc'不是内部或外部命令，也不是可运行的程序或批处理文件。

4.3　CDT 的下载和安装

目前 CDT 的最新版本为 CDT 2.0，能够支持 Eclipse 3.0，本书选用 CDT 1.2，请到 http://www.eclipse.org/cdt/上下载，该网页还有很多与 CDT 有关的资源，读者可以根据自身需求选择。

CDT 对于不同的平台有不同的版本，有 Win32 所有版本，Linux，Solaris QNX Neutrino，编者的系统是 Windows 2000，下载的文件是“cdt-full-1.2-win32.zip”，它包括了 Platform, Standard Make, Managed Make 三部分，是一个 CDT 的完整版。请读者根据自己的情况下载适合的版本。

CDT 的安装过程与 Eclipse 其他插件的安装过程相同，先将下载的文件解压，解压后，在文件夹 eclipse 下有两个文件夹 plugins 和 features。将这两个文件夹下的所有文件分别复制到 eclipse 安装路径下的 plugins 和 features 文件夹里。复制后启动 Eclipse，会出现如图 4-9 的对话框，询问你是否保留变更，请单击“完成（F）”。



图 4-9　设置更改对话框

之后，会询问是否要重启 Eclipse，如图 4-10 所示。选择“是（Y）”，Eclipse 重启。



图 4-10　选择“是（Y）”，Eclipse 重启

重新启动 Eclipse 后，选择“窗口”→“设定”，如图 4-11 所示出现了“C/C++”选项，则说明 CDT 插件安装成功。

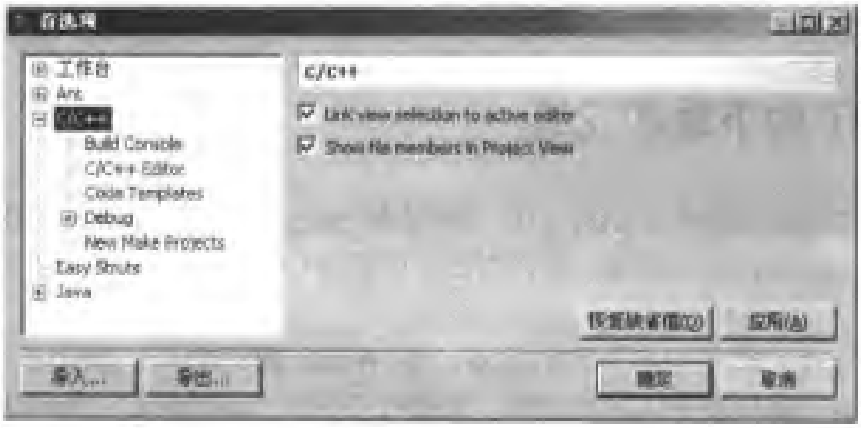


图 4-11　首选项中的“C/C++”选项

4.4　创建 C/C++工程

4.4.1　新建一个 C++工程

新建 C++工程可遵循以下步骤来进行：

（1）选择“文件”→“新建”→“项目”，在弹出的窗口中有 C 和 C++两个选项。

（2）选择“C++”。

（3）选择“Standard Make C++ Project”。如果选择“Managed Make C++ Project 选项，则可以自动生成 makefile 文件。

（4）选择“Next”，填入工程名“HelloWorld”。

（5）选择“Next”。在该窗口中：

- 选择“Binary Parser”项，在该项中选择“PE Windows Parser”。
- 选择“Make Builder”项。

 Builder Command 是用来描述创建时采用的创建语句，默认语句为“make”，因为安装了 MinGW，所以，将“Use Default”的选择去掉，并在文本框中填入“mingw32-make -f makefile”。

 Workbench Build Behavior 是用来表述创建时的动作行为的，其中：

 - Build on resource save(Auto Build)：在保存文件时创建。
 - Builder——在单击“Project”→“创建”时创建。
- Rebiuld——重建。

（6）选择“结束”。

这样，一个新的 C++工程就建立起来了，在这个工程中将会保存编写的源代码文件和工程配置文件。

4.4.2　新建.cpp 文件

现在，可以在刚刚建成的 C++工程下编写代码了。

（1）右击 HelloWorld，在弹出的下拉框中选择“新建（W）”→“文件”。

（2）在“文件名”中填入 main.cpp，然后单击“完成”。

（3）在文件 main.cpp 中输入如下代码：

```
#include <iostream>
using namespace std;
 int main () {
// Say Hello five times
    for (int index = 0; index < 5; ++index)
    cout << "HelloWorld!" << endl;
    char input = 'i';
    cout << "To exit, press 'm'" << endl;
    while(input != 'm') {
       cin  >> input;
       cout << "You just entered " << input
            << " you need to enter m to exit." << endl;
    }
    exit(0);
 }
```

（4）单击“文件”→“保存”。

编写完 C++源代码后，选择“project”→“创建”，在 C-Builder 中会显示出类似如下错误：

```
mingw32-make -f makefile clean all
mingw32-make: makefile: No such file or directory
mingw32-make: *** No rule to make target `makefile'.  Stop.
```

这是因为缺少 makefile 文件。创建一个 makefile 文件是为了使编译器知道如何构建执行文件。

4.4.3　新建 makefile 文件

为修改上面提到的错误，新建一个 makefile 文件，具体步骤如下：

（1）右击 HelloWorld，在弹出的下拉框中选择“新建（W）”→“文件”。

（2）在“文件名”中填入 makefile，然后单击“结束”。

（3）在“makefile”文件中输入如下语句：

```
main : main.o
        g++ -o main main.o -L D:\PG\MinGW\lib\gcc-lib\mingw32\3.2.3 -lstdc++
main.o : main.cpp
        g++ -ggdb -c main.cpp
all :
        ${MAKE} main
clean :
-del main.o
```

4.5 CDT IDE 功能

4.5.1 自定义 C/C++编辑器

用户可以在首选项页面中设置 C/C++编辑器的各种属性，设置方法可参照以下步骤进行：

（1）选择“Window”→“Preferences”。

（2）展开“C/C++”，选择“C/C++ Editor”，如图 4-12 所示。



图 4-12 首选项中的“C/C++ Editor”

在“C Editor Preferences”中，共有四个选项卡：“General”，“Annotations”，“Colors”，“Content Assist”（见图 4-12 中间上方所示）。

下面分别介绍各项中的主要内容。

① General 选项卡包括以下内容。

- “Displayed tab width”：显示的“Tab”键的长度。
- “Print margin column”：打印边距的长度。在实际工作中，调整该项的值可以提醒用户不要超出规定的长度，这样会避免超长代码的产生从而提高代码的易读性和

美观。

- "Insert space for tabs"："Tab"中是否含有空格。
- "Highlight matching brackets"：括号高亮度显示。如果选择该项，则当光标放在括号上时，该括号处于高度显示。
- "Show line numbers"：显示行号。如果选择该项，则编辑器中将显示代码区域每一行的行号。
- "Highlight current line"：高亮度显示当前行。
- "Show overview ruler"：显示外观尺。选择该项则编辑器的右侧多出一个竖条。
- "Show print margin"：显示打印边距。

② Annotations 选项卡（如图 4-13 所示）包括以下内容。

- Annotations Presentation Options：注释陈述选项，其中包括以下六项。
 - Problems：语法上出现问题地方会根据你描述的状态显示。
 - Warnings：系统提示注意的地方会根据你描述的状态显示。
 - Infos：描述的情况。
 - Tasks：任务。
 - Bookmarks：书签。设定的书签在外观尺的状态。
 - Search Results：检索结果。
- Show In Text：在文本中显示。
- Show In Overview Ruler：显示在外观尺中。
- Annotations Color：显示的颜色。



图 4-13　"C/C++ Editor"中的"Annotations"

③ Color 选项卡描述编辑器中各项的颜色（如图 4-14 所示）包括以下内容。

- Background Color：背景颜色。有两个选项，System Default 是系统默认颜色白色；Custom 是用户自定义的颜色，一般情况下，选择白色。
- Foreground：在代码区域中的情况，包括以下七项。

- Multi-line comment：多行注解。在C/C++中的多行注释可以放在“/**/”中，如果颜色选择蓝色则注释就是蓝色。
- Single-line comment：单行注释。在C/C++中的单行注释可以放在“//”后，如果颜色选择蓝色则注释就是蓝色。
- Keywords：关键字。
- Built-in types：嵌入类型的标识符，如include。
- Strings：字符串类型的变量值。
- Others：其他的设置。
- Task Tags：任务标签。

④ Content Assist 选项卡：表示代码辅助功能，其内容在后面的章节中详细介绍。



图 4-14 “C/C++ Editor”中的“Colors”

4.5.2 代码模版

写代码的时候通常会用到“if(){...}else{...}”、“while(){...}”等这样的语句，用户还可能用到自己编写的一块小代码或结构，把这样的语句或结构称为代码模版（Code Templates）。CDT 提供了生成、编辑、使用代码模版的功能。通过代码模版不仅可以快

速、规范地书写代码，而且可以避免一些不该发生的错误。

下面介绍 CDT 提供的代码模版的编辑、生成等管理代码模版功能。

选择“Window”→“Preferences”，打开“C/C++”的下拉框，找到“Code Templates”，如图 4-15 所示。在该窗口可以实现代码模版的生成（New）、编辑（Edit）、删除（Remove）、导入（Import）、导出（Export）等功能。下面分别介绍这些功能。



图 4-15　“C/C++”中的“Code Templates”

1．生成代码模版

见图 4-15，单击“New...”，出现如图 4-16 的窗口。其中：

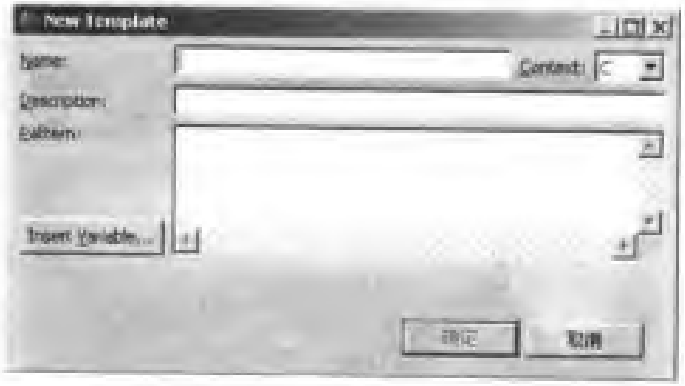


图 4-16　新代码模版

Name——代码模版的名字。

Context——可以选择 C 或 C++。

Description——内容描述。

Pattern——代码模版的模式。

Insert Variable——插入变量。比如插入日期变量 date。

下面举个例子来说明（在 4.5.3 小节中会用到这个代码模版），请按照如下给出的内容填写图 4-17 显示的窗口。

```
Name : test
Description : Templates test
Pattern :
    test {
        //${date}
        ${testContext}
    }
```

请注意代码模版中的变量格式：${变量}。



图 4-17 “代码模版” test 的设置信息

2. 编辑代码模版

如图 4-18 所示，选择“author”之后，单击“Edit...”，出现如图 4-19 所示的窗口。试着修改一下该代码模版。将“Pattern”中的内容修改为：author ${LiuChuanYi} ${date}。修改后的内容如图 4-19 所示，单击“确定”后，编辑工作就完成了。



图 4-18 编辑“代码模版” author

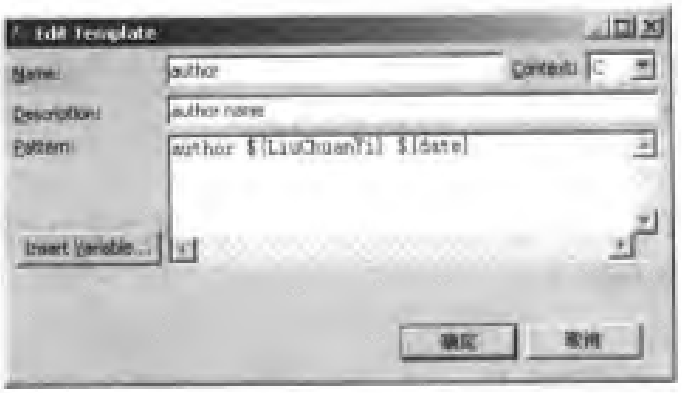


图 4-19　将“Pattern”中的内容修改为：author ${LiuChuanYi} ${date}

3. 删除代码模版

如图 4-20 所示，任意选择一个代码模版，然后单击“Remove”。

4. 导入代码模版

现在来做一个导入代码模版的例子，在 D 盘下建一个 templateImporTest.xml 文件，在文件中写入以下内容：

```
<?xml version="1.0" encoding="UTF-8" ?>
<templates>
<template name="test_01" description="test01 name" context="C++"
enabled="true">test_01 ${user}2</template>
<template name="test_02" description="test02 name" context="C"
enabled="true">test_02 ${user}1</template>
</templates>
```

保存文件后，打开如图 4-20 所示的窗口，选择“Import...”，并找到 D 盘下 templateImporTest.xml 文件，并将该文件导入。这时会发现多出两个代码模版：test_01 和 test_02，即图 4-20 中用虚线标记的部分。



图 4-20　多出两个代码模版 test_01 与 test_02

5. 导出代码模版

如图 4-20 所示，选中一个代码模版后，单击“Export...”，则该代码模版的信息就被导出到一个.xml 文件中，系统默认为 templates.xml。

6. 全部导出

如图 4-20 所示，单击“Export All...”，则所有代码模版就被导出到一个.xml 文件中，系统默认为 templates.xml。

7. 全部可用

如图 4-20 所示，单击“Enable All”，则显示的代码模版全部可以用。

8. 全部不可用

如图 4-20 所示，单击“Disable All”，则显示的代码模版全部不可以用。

4.5.3 代码辅助功能

现在通过新建一个小例子来向读者介绍 CDT 的代码辅助功能。

1. 新建一个类

（1）新建一个名为“Code_Test”的类。用右击工程“HelloWorld”→“新建（W）”→“Class”，之后，出现“New C++Class”对话框，如图 4-21 所示。



图 4-21 新建“Code_Test”类

（2）单击“完成（F）”后，生成的代码如下：

```
#include "Code_Test.h"

Code_Test::Code_Test()
```

```
{ }
Code_Test::~Code_Test()
{ }
```

2. 代码援助

下面通过 Code_Test 和 main 来对 CDT 插件提供的代码辅助功能进行介绍。

（1）选择“窗口（W）”→“首选项（P）”。

（2）展开“C/C++”→选择“C/C++ Editor”→“Content Assist”，如图 4-22 所示。



图 4-22　“C/C++ Editor”中的“Content Assist”

其中：

- “Search entire project for completion proposals”选项——为完成用户要求查找整个工程。

 我们对比一下两种情况，首先是不选择该项的情况。

 - 在 main.cpp 文件中加入一个 get_main 方法并保存，不用加入方法体。
 - 在 Code_Test.cpp 中加入两个方法：get_codeTest_01 和 get_codeTest_02 并保存，都不用加入方法体。
 - 在 Code_Test.cpp 中的构造器 Code_Test 中写入“get”。按“Alt”+“/”，则会有两个待选项出现，如图 4-23 所示。可以选择这两项中的任意一个，而且不

用写出完整的方法名。

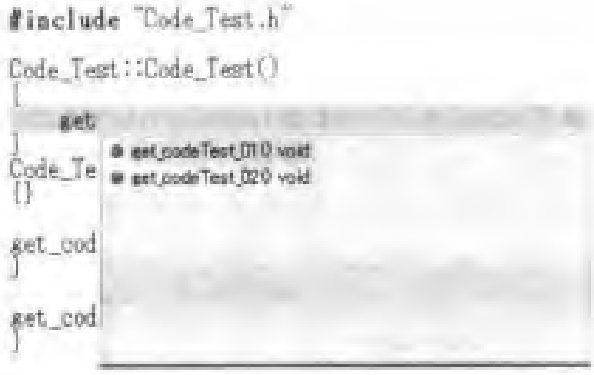


图 4-23 出现两个待选项

如果选中 Search entire project for completion proposals 项，则重复上述三步，效果如图 4-24 所示。

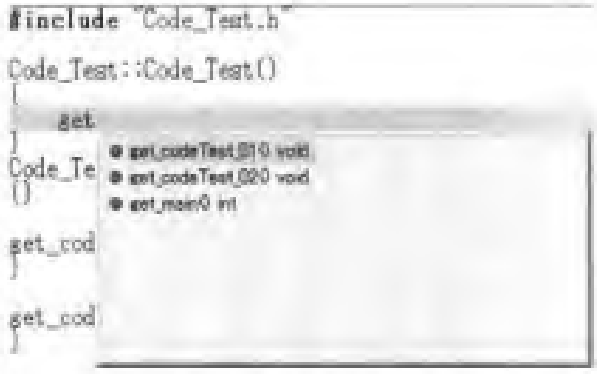


图 4-24 新的待选项出现

- "Insert single proposals automatically" 选项——自动插入单条需求。
 如果不选中该选项，按照以下步骤操作：
 - 将 get_codeTest_01 和 get_codeTest_02 方法删除。
 - 在 Code_Test.cpp 中的构造器 Code_Test 中写入 "get"。按 "Ctrl" + "/"，出现的情况如图 4-25 所示。

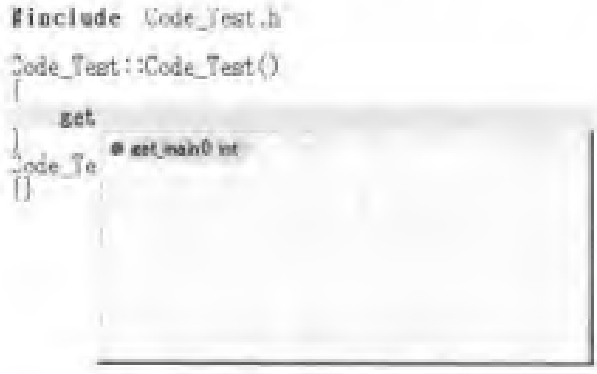


图 4-25 只有一个待选项出现了

如果选中该项，执行上述步骤②，则会发现 get_main 方法自动将全称添加好了。

- Enable auto activation——当输入第一个字母，会自动打开代码援助。
- Auto activation delay——自动激活的延迟时间。
- Auto activation triggers for C——在延迟时间内调用代码辅助功能。
- Background for completion proposates——为文本对话框设置后台颜色。
- Foreground for completion Proposates——为文本对话框设置前台颜色。

4.5.4　生成本地历史记录

在“Code_Test.cpp”上右击鼠标→“替换为（L）”→“本地历史记录（L）...”，则“从本地历史记录中替换”窗口弹出，如图 4-26 所示。单击窗口中的“替换”按钮，现在的程序就会被替换为历史记录的程序。

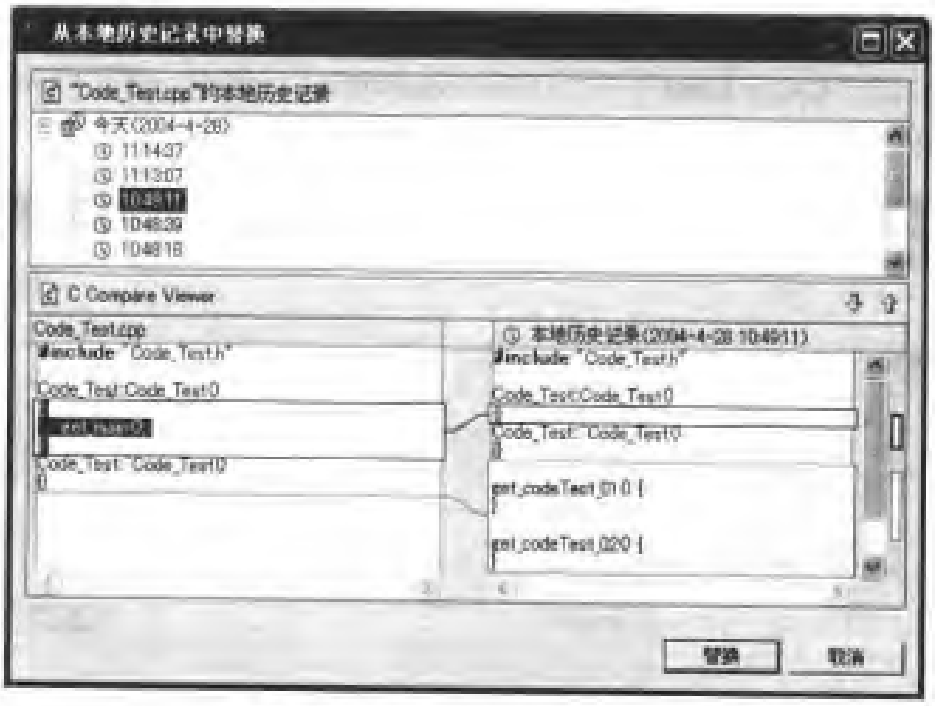


图 4-26　从“本地历史记录中替换”窗口

4.6　调试 C++工程

下面调试 HelloWorld 工程。

（1）切换到“C/C++ Development 透视图”，单击“运行（R）”→“调试（B）...”，则“调试”窗口弹出。如图 4-27 所示。



图 4-27 “调试”窗口

（2）双击“C/C++ Local”，在“C/C++ Application”一栏中，填写“main.exe”，如图 4-28 所示。

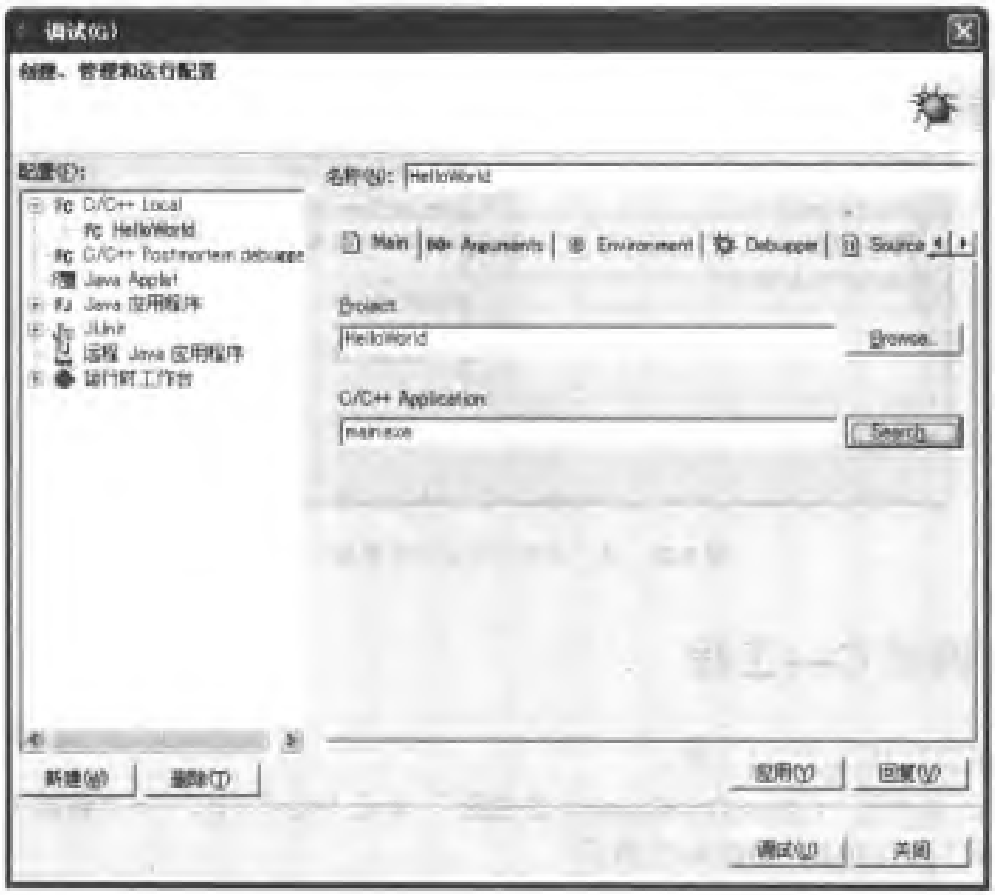


图 4-28 在“C/C++ Application”一栏中，填写“main.exe”

（3）单击“调试（U）”。

（4）单击“运行（R）”→继续（M），则运行窗口如图 4-29 所示。



图 4-29　运行窗口

（5）输入字母“m”，按下“Enter”键后，调试结束。

本章小结

Eclipse 协会称 CDT 是向集成的跨平台开源 IDE 迈出的重要一步，可以看出 CDT 是 Eclipse 一个重要的发展方面，CDT 在功能上、性能上会不断地被完善。

本章讲解了 CDT 插件及其安装，以及如何利用 CDT 插件创建 C/C++工程。希望读者通过这一章的阅读能够体会到 Eclipse 的强大，并使其在项目开发中成为自己的有力工具。

第三篇

Eclipse 测试工具

PART 3

第 5 章　Eclipse 与 JUnit

第 5 章　Eclipse 与 JUnit

Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台。就其本身而言，它只是一个框架和一组服务，用于通过插件组合构建开发环境。Eclipse 附带了一个标准的插件集，包括 Java 开发工具（Java Development Tools，JDT）。Eclipse 平台为插件开发提供一组功能强大的服务和 API。它使来自完全不同的供应商的工具之间的集成变得平滑，为不同类型的开发工作创建了一个无缝的环境。其中就包括著名的工程测试框架工具——JUnit。

Eclipse IDE 极好地集成了 JUnit，它有自己的测试套件，使得测试环境更加宽松，便于操作和管理，在本章中，将详细介绍有关软件测试的基础知识、JUnit 的原理和在 Eclipse 中使用 JUnit 的方法。

本章将要介绍的内容如下：

- 软件测试基础。介绍软件测试的基本概念、重要性以及测试的重点和方法介绍。
- JUnit 简介。介绍 JUnit 的特点，以及使用 JUnit 的优点。
- 在 Eclipse 中设置 JUnit。介绍怎样在 Eclipse 中整合 JUnit 测试程序。
- 用 Eclipse 结合 JUnit 进行单元测试。通过实例，具体讲解 JUnit 在 Eclipse 中是怎样完成测试任务的。

5.1　软件测试基础

软件测试就是在软件投入使用前，对软件需求分析、设计规格说明和编码的最后审查，这是软件质量保证的关键步骤。软件测试是为了发现错误而执行程序的过程。软件测试在软件开发过程中横跨两个阶段：通常在编写出每一个模块之后就对它做必要的测试（称为单元测试）。编码和单元测试属于软件开发过程中的同一个阶段。在结束这个阶段后对软件系统还要进行各种综合测试，这是软件开发过程中的另一个独立阶段，即测试阶段。

5.1.1　软件测试的目的

引用 Grenford J. Myers 在《The Art of Software Testing》一书中的观点，软件测试的目的就是：

（1）软件测试是为了发现错误而执行程序的过程；

（2）测试是为了证明程序有错，而不是证明程序无错误；

（3）一个好的测试用例是在于它能发现至今未发现的错误；

（4）一个成功的测试是发现了至今未发现的错误的测试。

这种观点提醒人们测试要以寻找错误为目的，而不是为了演示软件的正确功能。但是，测试并不仅仅是为了要找出错误，通过分析错误产生的原因和错误的分布特征，可以帮助项目管理者发现设计开发软件过程的缺陷，以便改进。同时，这种分析也能帮助用户设计出有针对性的检测方法，改善测试的有效性。其次，没有发现错误的测试也是有价值的，完整的测试是评定测试质量的一种方法。

5.1.2 软件测试的原则

软件测试的原则包括以下几点：

（1）尽早地不间断地进行软件测试。

（2）测试实例应由测试输入数据和与之对应的预期输出结果两部分组成。

（3）整体测试工作应该由独立的专业的软件测试机构来完成。

（4）在设计测试用例时，应当包括合理的输入条件和不合理的输入条件。不合理的输入条件是指异常的、临界的，可能引起问题异变的输入条件。

（5）充分注意测试中的群集现象。经验表明，测试后程序残存的错误数目与该程序中以发现的错误数目或检错率成正比。应该对错误群集的程序段进行重点测试。

（6）严格执行测试计划，排除测试的随意性。测试计划应包括：所测软件的功能、输入和输出、测试内容、各项测试的进度安排、资源要求、测试资料、测试工具、测试用例的选择、测试的控制方法和过程、系统的组装方式、跟踪规则、调试规则、回归测试的规定，以及评价标准等。

（7）对每一个测试结果做全面的检查与核实。

（8）妥善保存测试计划、测试用例、出错统计和最终分析报告，为维护提供方便。

5.1.3 软件测试的对象

软件测试不等于程序测试。软件测试应该贯穿整个软件定义与整个开发期间。因此“需求分析、概要设计、详细设计以及程序编码”等各阶段所得到的文档，包括需求规格说明、概要设计规格说明、详细设计规格说明以及源程序，都是软件测试的对象。

在对“需求理解与表达的正确性、设计与表达的正确性、实现的正确性以及运行正确性”的验证中，任何一个环节发生了问题都可能在软件测试中表现出来。

确认是一系列的活动和过程，其目的是想证实在一个给定的外部环境中软件的逻辑正确性。它包括需求规格说明的确认和程序的确认，而程序的确认又分为静态确认与动态确认。静态确认一般不在计算机上实行执行程序，而是通过人工分析或者程序正确性证明来确认程序的正确性，也就是通常所说的代码走读（code review）。动态确认主要是通过动态分析和程序测试来检查程序的执行状态，以确认程序是否有问题。

验证则试图证明在软件生存期间的各个阶段，以及各阶段间的逻辑协调性、完备性

和正确性。验证对象包括：用户要求、需求说明书、设计说明书、源程序及运行结果。

5.1.4　软件测试的种类

软件测试种类包括：

1．单元测试（unit test）

单元测试是为检查个别模块（比如 classes 类）服务的。如果对象需要访问外部的数据源（比如数据库），就需要通过一些模拟的对象（MOCK object）来模拟数据库，但这也只有在真实环境的数据与测试环境不同的时候——比如测试环境里面没有真实的数据库，才需要 MOCK Object。

2．用户测试（customer's test）

这是功能的、系统的并且认可的测试。系统中所有的行为检查都做为一个整体。在极限编程理论中，这些测试是由用户编写的，它给出测试案例提纲。

3．集成测试（itegration tests）

这些测试像是在用户测试和单元测试之间的十字路口。集成测试帮助程序测试几个级别中交互。Mock object 不会出现在集成测试中，它会增加测试时间。同样，集成测试也经常需要存在的特定的测试环境，比如在数据库中放一些测试数据。集成测试也可以使用外部的工具。Cactus 就是这样一个 J2EE 集成的工具包。解释这些测试已经超出了本书的范围，并且也需要详细的理论叙述，所以，读者仅需要知道这种测试存在就可以了。

4．开发者的测试（developer's test）

开发者的测试就是那些开发者校验整段代码、新加的代码、新加的函数。对于每个开发者而言，随时生成新的测试去检查代码是很重要的。组织这些测试和组织这些代码有着同样的重要性。

5.1.5　软件测试的过程

当软件设计工作完成后，就应该进行测试的准备工作了。一般来说，需要由熟悉整个系统设计的人员编写测试大纲和测试通过的细则、设计完整的测试用例，以便进行系统的全面测试。

软件测试一般分下面几个步骤。

1．编写测试式样书

测试人员要仔细阅读相关资料，全面熟悉系统，编写测试计划，设计测试用例，做好测试准备。

2．代码评审（code review）

通过静态分析（static analysis）和动态分析（dynamic analysis），对软件的源代码进

行研读，查找错误或收集一些度量数据。

3. 单元测试

单元测试是针对软件设计的最小单位——程序模块，进行正确性检测的测试工作。目的在于发现每个模块的内部可能存在的错误。一般采用结构测试（白盒法）的用例，尽可能达到彻底测试，然后辅助以功能测试（黑盒法）的用例，使之对任何合理和不合理的输入都能鉴别和响应。高可靠性的模块是组成可靠系统的坚实基础。

4. 集成测试

集成测试是将模块按照设计要求组装起来同时进行测试，主要目标是发现与接口有关的问题。如数据穿过接口时可能丢失；一个模块与另一个模块可能有由于疏忽的问题而造成有害影响；把子功能组合起来可能没有产生预期的主功能；个别看起来是可以接受的误差可能会积累到不能接受的程度；全程数据结构可能有错误等。

5. 验收测试

验收测试的目的是向未来的用户表明系统能够像预定要求那样工作。经集成测试后，已经按照设计把所有的模块组装成一个完整的软件系统，接口错误也已经基本排除了，接着就应该进一步验证软件的有效性，这就是验收测试的任务，即软件的功能和性能如同用户所合理期待的那样。

经过上述这几项测试过程对软件进行测试后，软件就基本符合开发的要求了。测试结束后，就可以将软件交付给用户了。

5.1.6 软件测试的基本方法

软件测试的基本方法包括下述两种测试方法。

1. 黑盒测试

黑盒测试又称功能测试或数据驱动测试，它是在已知产品所应具有的功能，通过测试来检测每个功能是否都能正常使用。这种测试是在软件的接口处进行的，测试人员不需要考虑程序内部的逻辑结构和内部特性，只依据程序的需求说明书，检查程序的功能是否符合它的功能说明。

黑盒测试主要为了发现以下问题：

- 是否有不正确的功能或者遗漏了某些功能；
- 在接口上，输入能否正确接受，并且能否输出正确的结果；
- 性能上是否能够满足要求；
- 是否有初始化或者终止性错误。

2. 白盒测试

白盒测试又称结构测试或逻辑驱动测试，它是知道产品内部工作过程，可通过测试来检测产品内部动作是否按照规格说明书的规定正常进行，按照程序内部的结构测试程序，检验程序中的每条通路是否都有能按预定要求正确工作，而不顾它的功能。白盒测

试的主要方法有逻辑驱动、基路测试等，主要用于软件验证。

软件人员通过白盒测试方法，主要对程序进行如下检查：

- 对程序模块的所有独立的执行路径至少测试一次；
- 对所有的逻辑判定，取“真”与“假”的两种情况都要至少测试一次；
- 在循环边界和运行界限内执行循环体；
- 测试内部数据结构的有效性，等等。

测试作为软件开发的重要环节，越来越受到开发人员的重视。随着软件开发规模的增加、复杂程度的提高，开展软件测试工作的重要性越来越明显。完整全面的测试是提高软件产品品质必不可少的。

5.2　JUnit 简介

JUnit 是一个开放源代码的 Java 测试框架，最初是由 Erich Gamma（Gang of Four，四人帮之一）和 Kent Beck（极限编程 XP 和重构 Refactor 的先驱之一）开发的，用在编写和运行可重复的测试上。它是用于单元测试框架体系 xUnit 的一个实例（用 Java 语言开发），包括以下特性：

- 用于测试期望结果的断言（assertion）；
- 用于共享共同测试数据的测试工具；
- 用于方便的组织和运行测试的测试套件；
- 图形和文本的测试运行器。

5.2.1　为什么使用 JUnit

JUnit 是在极限编程和重构中被极力推荐使用的工具，因为在实现自动单元测试的情况下可以大大提高开发的效率，但是实际上编写测试代码也是需要耗费很多的时间和精力的。使用这个工具，有以下好处：

1．利于极限编程

对于极限编程而言，要求在编写代码之前先写测试，这样可以强制你在写代码之前好好的思考代码（方法）的功能和逻辑，否则编写的代码很不稳定，那么你需要同时维护测试代码和实际代码，这个工作量就会大大增加。因此在极限编程中，基本过程是这样的：构思→编写测试代码→编写代码→测试，而且编写测试和编写代码都是增量式的，写一点测一点，在编写以后的代码中如果发现问题可以较快地追踪到问题的原因，以减小回归测试的纠错难度。

2．利于重构

对于重构而言，其好处和极限编程中是类似的，因为重构也是要求改一点测一点，减少回归测试造成的时间消耗。

3. 有断言功能

对于非以上两种情况，在开发的时候使用 JUnit 写一些适当的测试也是有必要的。因为一般情况下也是需要编写测试代码的，如果使用 JUnit，那么针对接口（方法）编写测试代码会减少以后的维护工作，例如以后对方法内部的修改（这个就是相当于重构的工作了）。另外一个好处就是 JUnit 有断言功能，如果测试结果不通过会告诉开发者是哪项测试不通过，而以前的一般做法是写一些测试代码看其输出结果，然后再由开发者自己来判断结果是否正确。使用 JUnit 的好处就是这个结果是否正确的判断是它来完成的，开发者只需要看其报告结果是否正确就可以了，因而会大大提高效率。使用 JUnit 实际上在提升程序码的质量的同时，仍允许编程者快速地编写程序。只要开始使用 JUnit，就会开始注意到介于程序代码及测试之间有一个强大的结合力。

以下列出的是使用 JUnit 的具体理由：

（1）在提升程序码的质量时 JUnit 测试仍允许更快速地编写程序。虽然听起来似乎不是很直观，但的确是这样的，这是事实。当使用 JUnit 编写测试，将花更少的时间除 Bug，同时对程序码的改变更具有信心。这样会让开发者更积极重整代码并增加新的功能。没有测试，对于重整及增加新功能会变得没有什么信心。因为不知道有什么东西会破坏生成的结果。采用一个综合的测试系列，可以在改变程序码之后快速地执行多个测试，并对于变动并未破坏任何东西感到有信心。在执行测试时如果发现 Bug，原始码仍然清楚地在脑中，因此很容易找到 Bug。在 JUnit 中编写的测试帮助开发者以一种极大的步伐编写程序及快速的找到缺点。

（2）JUnit 非常简单。编写测试代码应该很简单，这是必要的！如果编写测试太复杂或者太耗时间，便无法要求程序员编写测试程序。使用 JUnit 可以快速的编写测试程序，同时检测代码，并且逐步随着代码的增长增加测试。只要写了一些测试，想要快速并频繁的执行测试而不至于中断建立设计及开发程序便可能成为现实。使用 JUnit 执行测试就像编译的代码那么简单。事实上，应该在执行编译的同时也执行测试。编译检测程序代码的语法，而测试则检查代码的完整性。

（3）JUnit 测试检验其结果并提供立即的回馈。如果以人工对比测试的期望和实际结果，那么测试就不是一件“好玩”的事情了，而且这样会让编程速度慢下来。JUnit 测试可以自动执行并且检查它们自己的结果。当执行测试时，可以获得简单且立即的回馈，比如测试是通过或失败。这样编程者就不再需要人工检查测试结果的报告了。

（4）JUnit 可以合成一个测试系列的阶级架构。JUnit 可以把测试组织成测试系列。这个测试系列可以包含其他的测试或测试系列。JUnit 测试的合成行为允许组合多个测试并自动的回归，从头到尾测试整个测试系列。也可以执行测试系列界级架构中任何一层的测试。

（5）JUnit 测试的“低成本”性。使用 JUnit 测试框架编写一个测试就像写一个方法一样简单。测试是检验程序码并定义期望的过程。测试框架提供自动执行测试的环境，并且这个环境将成为其他测试集合的一部分。少量的测试投资将持续让开发者从时间及质量中获得回收。

（6）JUnit 测试提升软件的稳定性。写的测试越少，程序变得越不稳定。JUnit 测试

时的软件稳定并逐步积累信心，因为任何变动都不会造成连锁效应而漫及整个软件。

（7）JUnit 测试是开发者的测试。JUnit 测试是高度区域性测试，用以改善开发者的生产力及代码质量。与功能测试将系统看作一个黑箱进而确定软件整体的工作性为主不同，单元测试是由内而外，测试系统基础的建构区块。开发者编写并拥有 JUnit 测试。每当一个开发反复完成，这个测试便打包成为交付软件的一部分。

（8）JUnit 测试是以 Java 写成的。使用 Java 测试 Java 软件，形成一个介于测试及代码间的无缝边界。在测试的控制下，测试变成整个软件的扩充，与此同时代码可以被重构。Java 编译器的单元测试静态语法检查可以帮助测试程序并且确认遵守软件界面的约定。

（9）JUnit 是免费的。JUnit 是开源项目，源代码开放，属于免费的工具。

5.2.2　JUnit API 介绍

Junit.framework 包里包含开发人员编写 JUnit 测试所需要的所有类。其中，TestCase 类是这个包的核心部分，开发人员从这个类继承，建立自己的自动测试单元，其他的类是支持 TestCase 类，它们的作用是收集结果，聚合多个测试类成为测试系列（test suite），以及提供验证（assertion）。Test 是这个包里另一个关键的类，它定义 TestCase 及 TestSuites 共同的约定（contract），同时提供与框架连接（hook），以便扩充使用。

表 5-1 中列出了测试中常用的断言类别。

表 5-1　测试中常用的断言类别

断言方法	描　述
assertEquals	比较两个基本类型或对象是否相等
assertTrue	对布尔值求值，看是否为 true
assertFalse	对布尔值求值，看是否为 false
assertNull	检查对象是否为 null
assertNotNull	检查对象是否不为 null
assertSame	检查两个对象是否为同一实例
assertNotSame	检查两个对象是否不为同一实例

5.3　在 Eclipse 中设置 JUnit

在 Eclipse 中已经事先集成了 JUnit 插件，因此可以在 Eclipse 中直接运行测试程序，而不需要在额外地进行安装。尽管如此，仍然需要预先在 Eclipse 中设置 JUnit，而进行 JUnit 设置的关键便是需要定义“类路径变量”。接下来，演示一下如何定义“类路径变量”。

（1）选择“窗口”菜单，选中“首选项”并单击，如图 5-1 所示。



图 5-1　在首选项中设置 JUnit 插件

（2）在弹出的首选项对话框左边的下拉菜单中，打开 Java 菜单，选中并单击“类路径变量”，如图 5-2 所示。

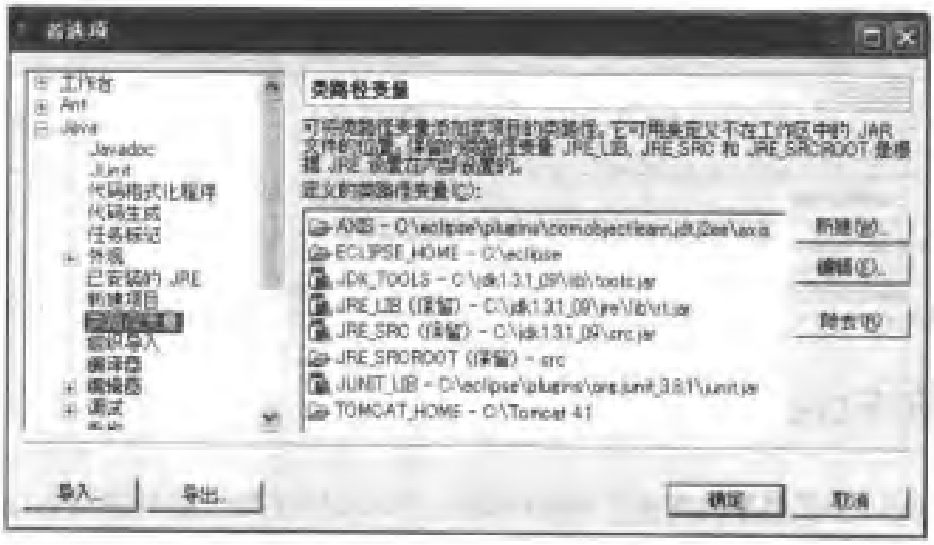


图 5-2　类路径变量

（3）单击“新建”按钮，在弹出的新建变量条目对话框里输入名称“JUNIT_LIB”和“junit.jar”的路径。JUnit 插件放在 Eclipse 的 plugins 文件夹下，如图 5-3 所示。



图 5-3　新建 JUNIT_LIB 变量

（4）单击“确定”按钮。图 5-4 显示了添加好的类路径变量。



图 5-4　添加完成画面

（5）单击“确定”按钮，完成添加类路径变量。

5.4　用 Eclipse 和 JUnit 进行单元测试

5.4.1　单项测试

下面通过一个简单的例子来介绍一下如何进行单元测试。

（1）新建一个名为“JUnitTest”的测试项目。新建项目的步骤可参考第 3 章内容。

（2）添加“junit.jar”库文件。在“JUnitTest”新建项目上点右键，选中“属性”，如图 5-5 所示。

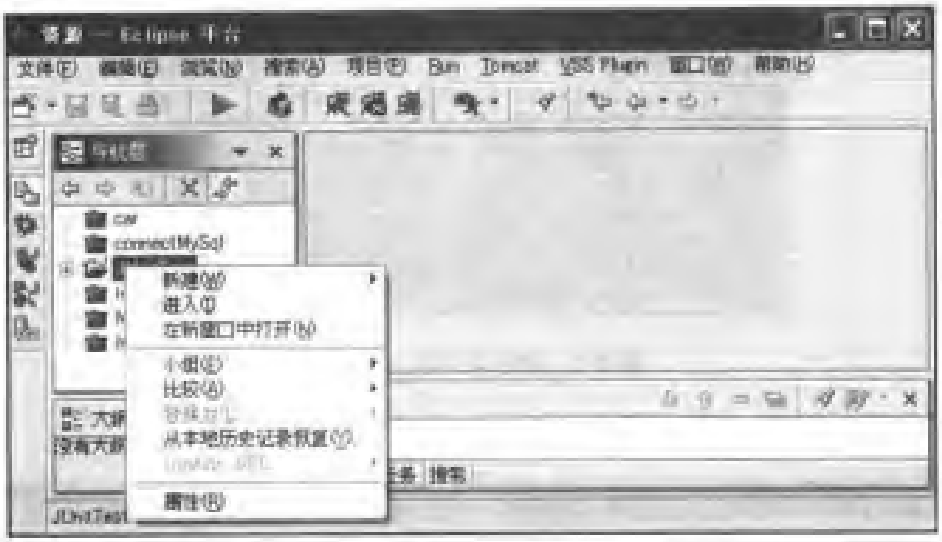


图 5-5　打开工程的属性框

在弹出的属性对话框里选中左边的“Java 构建路径”，然后选中右边的“库”选项，如图 5-6 所示。



图 5-6　添加变量

单击“添加变量”按钮，在弹出的“新的变量类路径条目”对话框中选中刚才添加的“junit.jar”，然后点“确定”，如图 5-7 所示。

（3）新建名为“StringUtil”的 Java 类，如图 5-8 所示。

（4）为“StringUtil.java”建测试类。

在资源管理器中右击“StringUtil.java”→“新建”→“其它”，如图 5-9 所示。

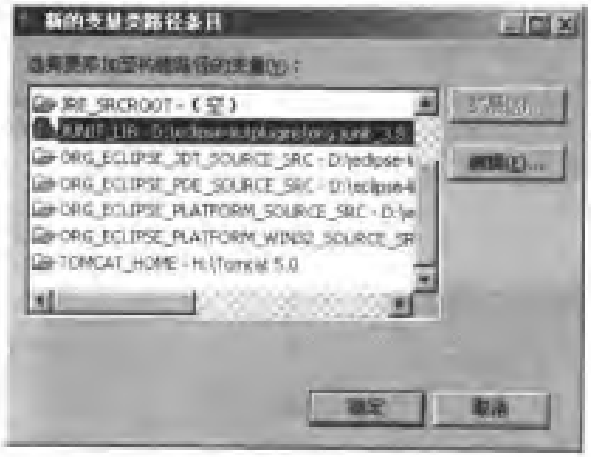


图 5-7　选择 JUNIT_LIB



图 5-8　新建 Java 类

在弹出来的新建对话框中选中左边 Java 菜单下的“JUnit”，然后选择右面的“测试案例”，单击“下一步”，如图 5-10 所示。

将多选框全部选中，在下拉菜单中选择图形化界面（swing ui）显示结果，如图 5-11 所示。

单击“完成”，至此就建了一个为名“StringUtilTest.java”的测试类。



图 5-9　新建测试类



图 5-10　选择 JUnit 测试案例

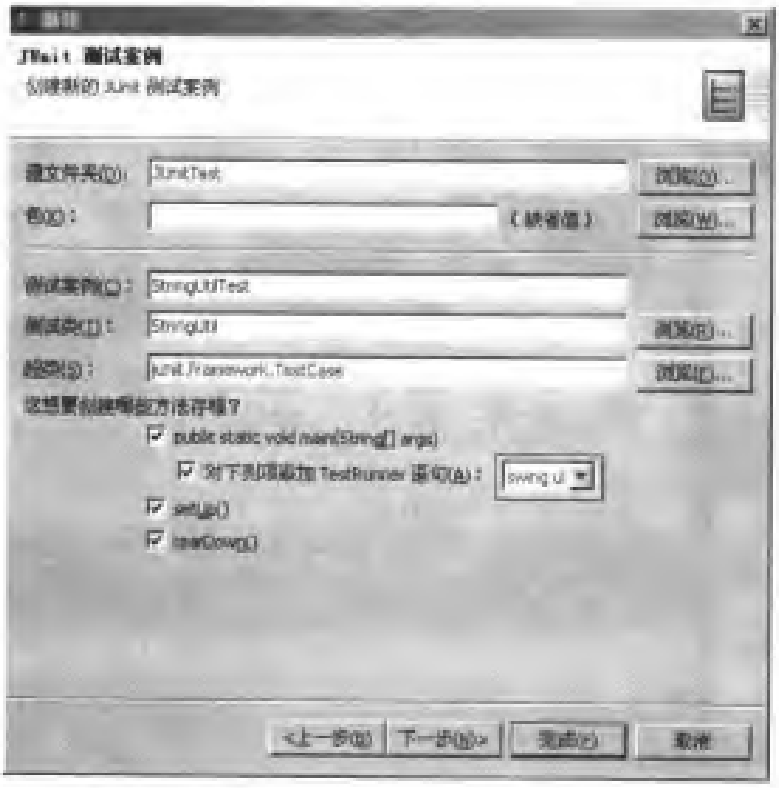


图 5-11　新建 JUnit 测试案例

（5）编辑测试类。

下面编辑测试程序。为了测试，先在被测试类“StringUtil.java”里加一个 split()方法，让它的返回值是 null，然后在测试程序“StringUtilTest.java”里加一个 testSplit()方法，语句如下：

```
public void testSplit() {
    String str = "a,b,c";
    String[] result = StringUtil.split(str, ",");
    assertNotNull(result);
    assertEquals(3,result.length);
    assertEquals("a", result[0]);
    assertEquals("b", result[1]);
    assertEquals("c", result[2]);
}
```

（6）调试测试程序。

保存修改后的文件，在工具栏上选中调试工具，选择调试为“JUnit 测试”，如图 5-12 所示。

因为被测试类“StringUtil.java”中的 split()方法返回的是 null，所以运行出来的结果是失败的画面，画面显示一条红杠，如图 5-13 所示。

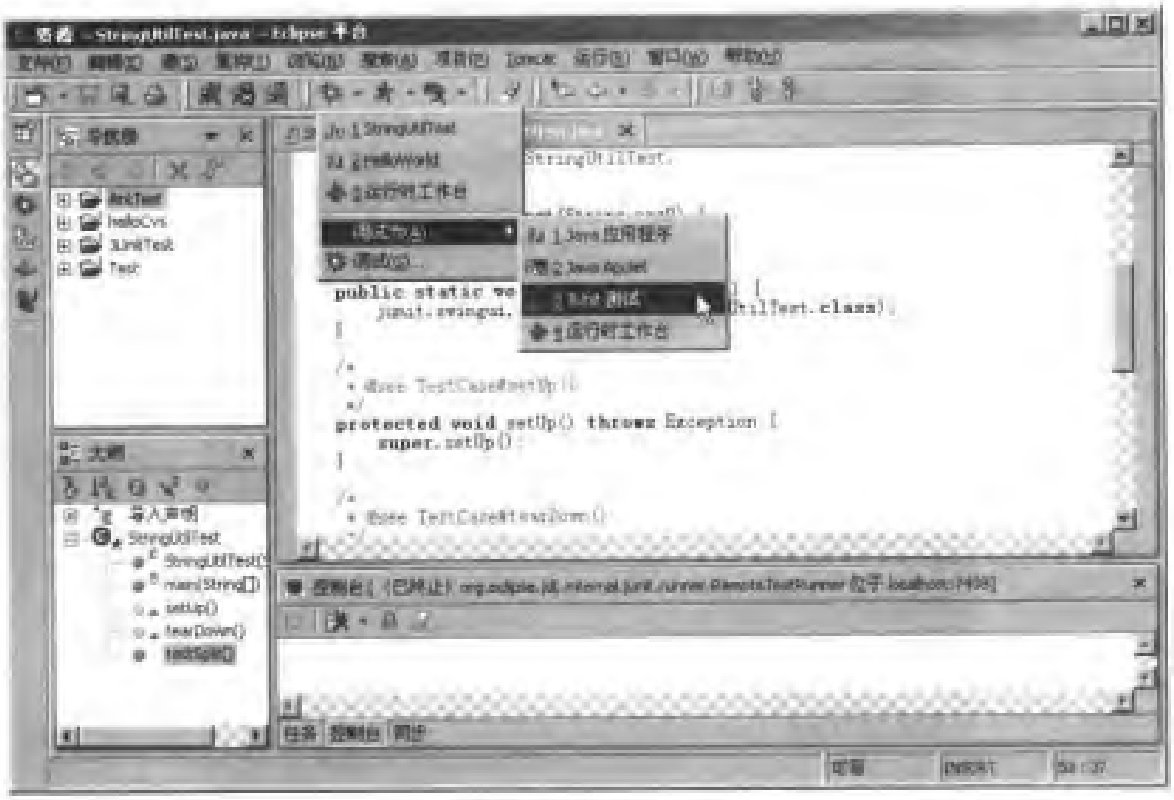


图 5-12　进行 JUnit 测试

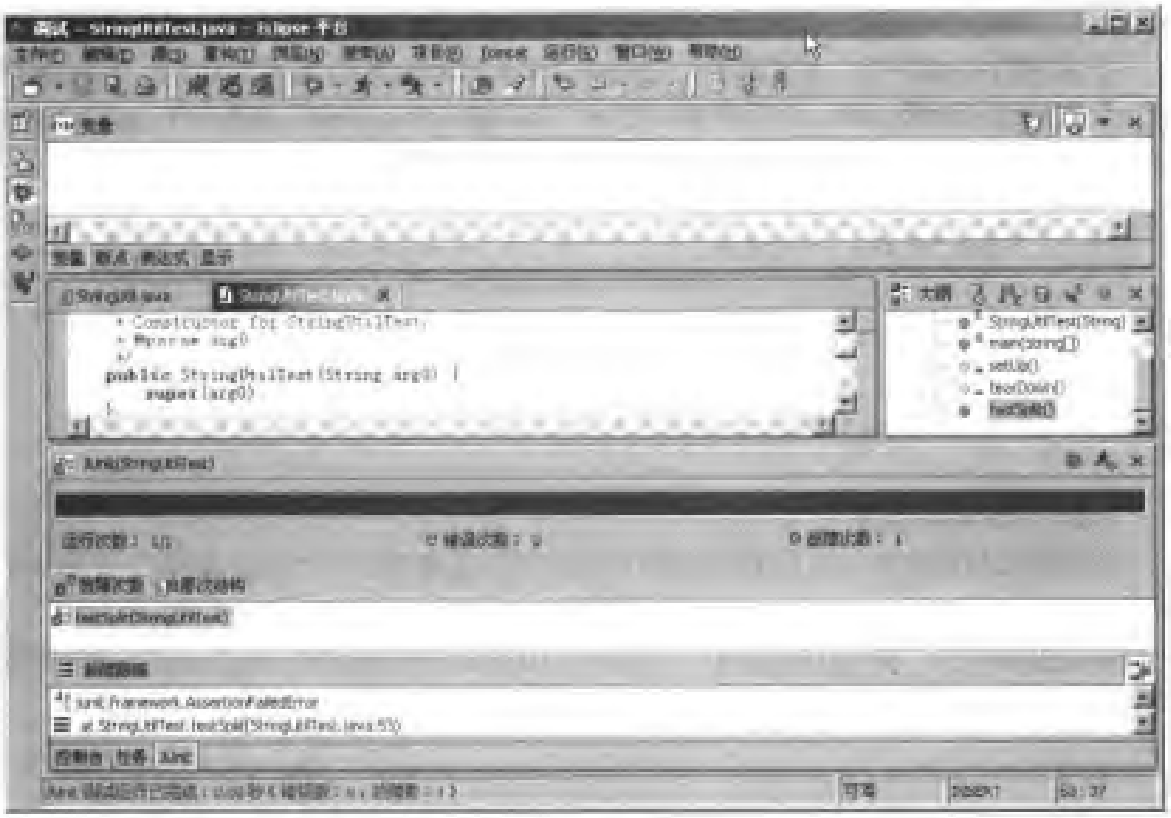


图 5-13　测试结果

（7）修改源程序。

现在修改“StringUtil.java”类的 split()方法，下面是 split()方法的全部内容：

```
public static String[] split(String str,String delim){
    StringTokenizer st = new StringTokenizer(str,delim);
    int length = st.countTokens();
    String[] result = new String[length];
    for (int i = 0; i < length; i++) {
        result[i] = st.nextToken();
    }
    return result;
}
```

注意：split()方法中用到 StringTokenizer 类时，需要导入：“import java.util.StringTokenizer;”修改保存后，再测试一次，结果是正确的，测试通过，画面显示一条绿杠，如图 5-14 所示

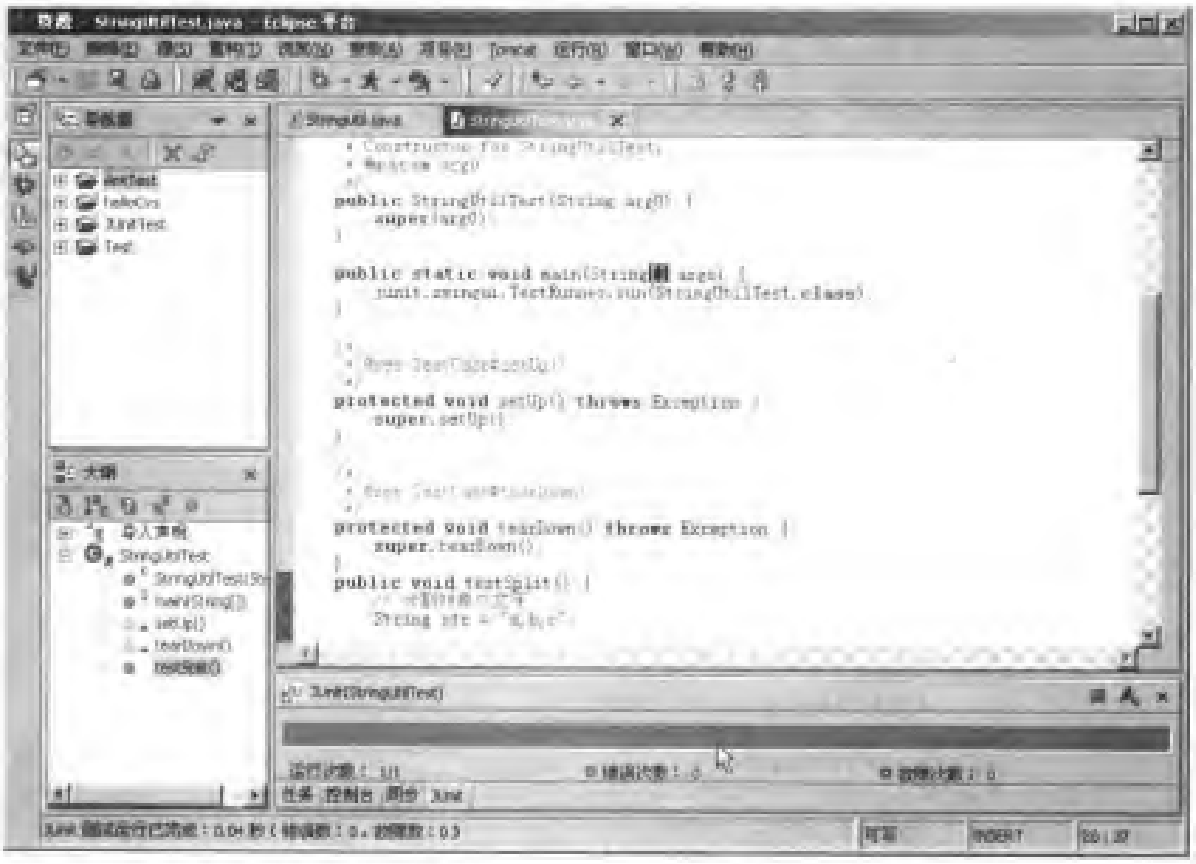


图 5-14　测试结果

5.4.2　多项测试

Eclipse 可以自动生成 JUnit 测试案例，所以使用起来很简单。JUnit 不但可以进行单个测试，还可以做多项测试，只要加几个以“test”开头的方法。下面介绍一个可以进行

多项测试的例子。

（1）修改一下“StringUtil.java”的 split()方法。修改后的方法如下所示：

```
public static String[] split(String str, String delim) {
    final int delimLength = delim.length();
    int pos = 0;
    int index = 0;
    ArrayList list = new ArrayList();
    while((index = str.indexOf(delim,pos))!=-1){
        list.add(str.substring(pos,index));
        pos = index + delimLength;
    }
    list.add(str.substring(pos));
    return ((String[])list.toArray(new String[0]));
}
```

（2）在“StringUtilTest.java”类中加两个测试方法。

① testSplit2()方法：

```
public void testSplit2() {
String str = "a,,";
String[] result = StringUtil.split(str, ",");
assertNotNull(result);
assertEquals(3, result.length);
assertEquals("a", result[0]);
assertEquals("", result[1]);
assertEquals("", result[2]);
}
```

② testSplit3()方法：

```
public void testSplit3() {
    String str = null;
    String result[] = null;
    str = "A*B*C";
    result = StringUtil.split(str, "*");
    assertNotNull(result);
    assertEquals(3, result.length);
    assertEquals("A", result[0]);
    assertEquals("B", result[1]);
    assertEquals("C", result[2]);
    str = "123 [SEP] 456 [SEP] 789";
    result = StringUtil.split(str, " [SEP] ");
    assertNotNull(result);
    assertEquals(3, result.length);
    assertEquals("123", result[0]);
    assertEquals("456", result[1]);
    assertEquals("789", result[2]);
}
```

（3）运行测试类。保存编辑后的文件，再按前面的步骤运行测试类。三个测试方法都运行并检测了修改后的“StringUtil.java”的 split()方法。测试结果如图 5-15 所示。

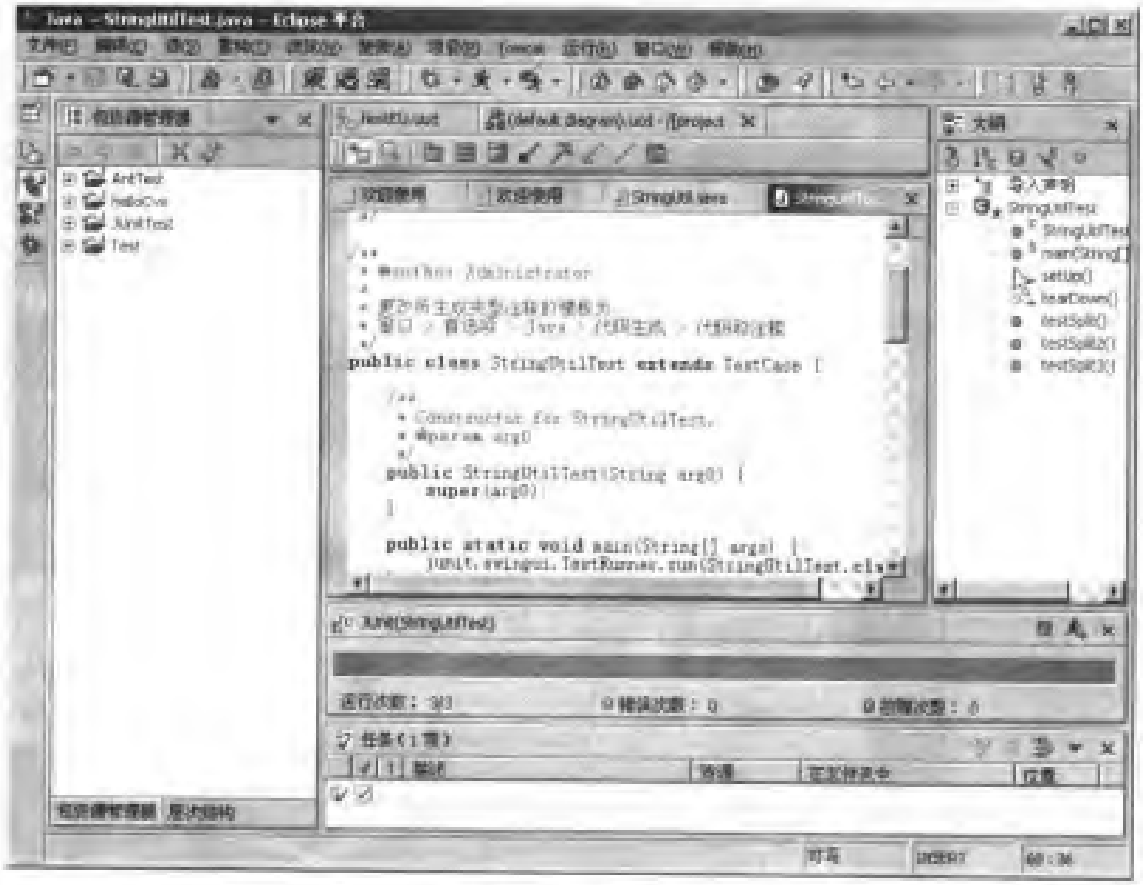


图 5-15 测试结果

5.4.3 多个测试类测试

Eclipse 结合 JUnit 还可以进行多个类的测试工作，下面演示用两个测试类进行测试。

（1）添加一个新的测试类“StringUtilTest2.java”，在该类中编写一个测试方法 testSplit1()。

testSplit1()方法的语句如下：

```
public void testSplit1(){
    String str = "a,,";
    String[] result = StringUtil.split(str, ",");
    assertNotNull(3, result[0]);
    assertEquals(2, result.length);
    assertEquals("a", result[0]);
    assertEquals("", result[1]);
    assertEquals("", result[2]);
}
```

（2）新建一个 JUnit 测试套件，如图 5-16 所示。

单击“下一步”将两个测试类都选中，如图 5-17 所示。

单击“完成”后，Eclipse 生成了一个测试套件类（AllTests），它将自动运行选中的两个测试案例，如图 5-18 所示，运行两个测试案例的方法。

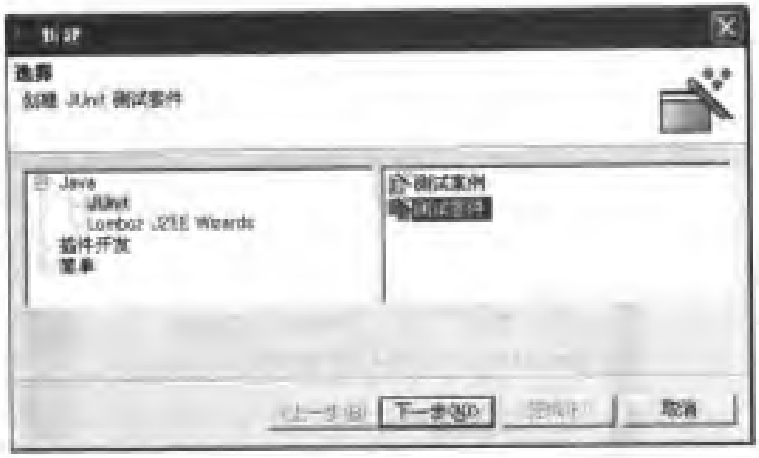


图 5-16　选择测试套件



图 5-17　选择测试类

（3）用调试工具运行测试套件类（AllTests）的结果，如图 5-19 所示。

Eclipse 结合 JUnit 测试程序方便快捷。上面介绍的例子是几种基本的测试形式，用 JUnit 还可以测试复杂的数据库程序。在下一节中，用一个实例详细介绍数据库程序的测试。

5.5　数据库程序的单元测试

数据库技术在软件设计中发挥着重要作用，无论是在电子商务或是现代的管理软件设计中都离不开数据库的支持。目前开发的软件项目大都跟数据库有着密切的关系，用户需要从数据库中存取更新数据，对于如此频繁复杂的操作，跟数据库打交道的程序测试更显得尤为重要。



图 5-18　增加的测试类



图 5-19　测试结果

下面，以一个连接 MySQL 数据库的程序为例，用 JUnit 对它进行测试。首先需要在本机上安装 MySQL 数据库，以及建数据库和加用户。具体操作细节请参阅第 10 章“数据库的构筑与连接”，在这里就不过多介绍了。现在要做的是编写测试类和被测试类。

按照上面的要求建数据库，加用户，建表操作之后，现在，数据库的状况如下：

- 用户名：mailAdmin；
- 数据库名：mldb；
- 表名：mlmember。

如果这几项都具备的话，就可以进行下一步了。用前面 5.4 节已经建好的项目“JUnitTest”，选中项目名，右击“属性”→“Java 构建路径”→“添加外部 Jar”，将 MySQL 驱动程序导入项目中。

1. 新建一个连接数据库的类（dbConn.java）

类的具体内容如下：

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class dbConn {
    public String getData() throws SQLException {
        String s_email = "";
        Connection con = null;
        Statement st = null;
        ResultSet rs = null;
        try {
           Class.forName("com.mysql.jdbc.Driver"); //加载数据库驱动程序
           con = DriverManager.getConnection("jdbc:mysql://localhost/mldb");
    //连接数据库
           st = con.createStatement();//创建一个 Statement 对象
             String sql = "";
           sql = "select * from mlmember where name=' Mike' ;// sql 查询语句
             rs = st.executeQuery(sql);  //创建一个 ResultSet 对象
             if(rs.next()) {
                 s_email = rs.getString(1); //处理从 DB 返回的数据
              }
        } catch (Exception e) {
          e.printStackTrace();
        }finally{
            if(rs != null) rs.close(); //关闭 ResultSet 对象
            if(st != null) st.close(); //关闭 Statement 对象
            if(con != null) con.close(); //关闭数据库连接
        }
        return s_email;
    }
}
```

2. 新建测试类

按照上一节创建测试程序的步骤，新建一个测试类“dbConnTest.java”。在复选框中，选中这两个方法：“setup()”、“tearDown()”。

3. 编辑测试类

测试类“dbConnTest”的内容如下：

```
import junit.framework.TestCase;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class dbConnTest extends TestCase {
    private Connection con;
    private Statement st;
    private ResultSet rs;

    /*
    *连接数据库
    */
    protected void setUp() throws Exception {
        //加载数据库驱动程序
        Class.forName("com.mysql.jdbc.Driver");
        //连接数据库
        con = DriverManager.getConnection("jdbc:mysql://localhost/mldb");
        //新建一个 Statement 对象
        st = con.createStatement();
    }

    /*
    * 测试 GetData 方法
    */
    public void testGetData() throws SQLException {
        dbConn o_dbConn = null;
        try {
          //实例化数据库连接
          o_dbConn = new dbConn();
          //向数据库中插测试数据
             setData();
        } catch (Exception e) {
          //异常处理
          assertEquals("1","0");
        }finally{
          //预期值与结果值比较
          assertEquals("邮件地址","Mike@sohu.com",o_dbConn.getData());
          //删除测试数据
          delData();
        }
    }

    /*
    * 插入测试数据
    */
    private void setData() throws SQLException{
        StringBuffer sql = new StringBuffer();
        sql.append("insert into mlmember values(' Mike@sohu.com' , ' Mike' );");
        st.executeUpdate(sql.toString());
    }

    /*
    * 删除测试数据
    */
    private void delData() throws SQLException{
        StringBuffer sql = new StringBuffer();
        sql.append("delete from mlmember");
        st.executeUpdate(sql.toString());
```

```
        }

        /*
        * 关闭数据库连接
        */
        protected void tearDown() throws Exception, SQLException {
            //关闭 ResultSet 对象
            if(rs != null) rs.close();
            //关闭 Statement 对象
            if(st != null) st.close();
            //关闭数据库连接
            if(con != null) con.close();
        }
    }
```

在这个测试程序里，有几个特殊的方法需要介绍一下。

（1）setUp()方法：setUp()方法是初始化方法，程序最先运行这个方法，所以把连接数据库放在这个方法里。

（2）tearDown()方法：tearDown()方法是程序最后运行的方法，主要用于清空对象。在本例里，用于关闭数据库连接。

（3）test...()方法：当运行测试时，JUnit 执行每个以“test”开头的方法；在执行每个测试方法之前执行 setUp()方法；在执行每个测试方法之后执行 tearDown()方法。

4．运行测试类

用 JUnit 执行一下测试类。比较预期的值和从数据库里取出来的值是否是一样的。运行结果是绿杠，说明数据库连接没有问题。

用 JUnit 测试和数据库相关的程序非常方便快捷，直接就可以看到从数据库取出得值与预期值对比的结果。如果想进一步深入了解 JUnit 用于测试的方法，请参考 JUnit 的 API。登录 JUnit 的官方网站（http://www.junit.org/index.htm），可以获得更多关于 JUnit 的最新信息及测试的技巧。

本章小结

本章首先介绍了软件测试的基础知识，包括软件测试目的、原则、测试对象及测试方法。然后着重介绍了 JUnit 测试工具，以及如何在 Eclipse 中结合 JUnit 测试工具进行单元测试。本章通过几个简单的实例，逐步地指导读者如何利用 JUnit 测试工具来编写测试程序。相信因为 JUnit 的方便快捷，程序员们不会再把编写测试程序当成一件苦差事了。

第四篇

Eclipse 资源管理工具

PART 4

第 6 章　Eclipse 环境下 Ant 插件的应用

第 7 章　使用 Eclipse 协同 CVS 进行版本管理

第 6 章　Eclipse 环境下 Ant 插件的应用

Eclipse 最大的优点是可以通过插件将许多实用的工具与 Eclipse 整合到一起，这样 Eclipse 就变得越来越强大。其中，Ant 就是一款不错的编译部署工具，在本章中将介绍 Ant 方面的知识以及在 Eclipse 环境下使用 Ant 插件。本章主要内容有：

- Ant 的简介及安装配置
- Ant 运行
- build.xml 语法详解
- 如何使用 Ant 插件部署
- Ant 与 JUnit 结合进行单元测试

6.1　Ant 简介

可能有些读者并不了解什么是 Ant，以及如何使用它。但只要使用过 Linux 系统或者 UNIX 系统，应该知道 make 这个命令，当编译 Linux 内核及一些软件的源程序时，经常需要在控制台下输入 make 命令。make 命令其实就是一个项目管理工具，而 Ant 所实现的功能与 make 差不多，它也是个项目管理工具。既然已经有了 make，还有诸如 gnumake，nmake，jam 以及其他的编译工具，为什么还要这种新的编译工具呢？因为上面提到的这些 build 工具有缺陷，其缺点是不能跨平台工作。类似于 make 的工具本质上是基于 shell 语言的，它们的计算依赖关系，然后执行命令，这就意味着可以通过使用操作系统特有的或编写新的 shell 程序扩展该工具。然而，这也意味着这个工具只能限制在特定的操作系统上使用。Ant 就没有这个缺陷。

Ant 是 apache 软件基金会 jakarta 项目中的一个子项目，它具有以下优点。

（1）跨平台性。Ant 是用 Java 语言编写的，因此具有很好的跨平台性。它可以在 Windows, Linux, UNIX 等各种环境下运行。不受平台差异影响。

（2）使用方便，操作简单。Ant 由一些内置任务（task）和可选择的任务组成（当然还可以编写自己的任务）。使用 make 时，需要写一个 Makefile 文件，而使用 Ant 时则需要写一个 build.xml 文件。由于采用 xml 的语法，通过调用 target 树，就可执行各种 task。每个 task 由一个实现了特定 Task 接口对象来运行。所以 build.xml 文件很容易书写和维护，且结构很清晰，而不像 Makefile 文件有那么多的限制（例如，在 tab 符号前有一个空格的话，命令就不会执行）。

（3）Ant 可以集成到开发环境中。由于 Ant 的跨平台性，以及操作简单的特点。它很容易的集成到一些开发环境中（例如 Eclipse, visual age, jbuilder, netbeans）。

（4）Ant 是开放源代码软件。Ant 是开源项目，不需要投入购买的资金。没有版权问题，可以到 Ant 网站（http://ant.apache.org/）下载使用。

6.2 Ant 安装和配置

因为 Ant 是一个 Open Source 的软件，所以有两种安装 Ant 的方法：一种是用已经编译好的二进制文件安装，另一种是自己编译 Ant 源代码。

下面分别介绍一下这两种安装方法。首先到 Ant 网站（http://ant.apache.org/）下载最新的 Ant 版本。二进制形式的 Ant 可以从（http://ant.apache.org/bindownload.cgi）下载；如果希望能自己编译 Ant，可以从（http://ant.apache.org/srcdownload.cgi）下载源代码文件。目前 Ant 最新发布的正式稳定版本是 1.6。如果想用最新的版本，可以从 Ant CVS 仓库（http://ant.apache.org/cvs.html)下载最新的源代码文件。

6.2.1 安装 Ant

先介绍一下二进制版安装 Ant。它相对于用源代码自己编译的安装方法简单很多。二进制版的 Ant 包括四个目录：bin，docs，etc 和 lib。只有 bin 目录和 lib 目录是运行 Ant 所必需的，docs 里主要是 Ant 的使用说明文档，etc 目录里放 xsl 文件。

首先，新建一个 Ant 的工作目录，将下载的二进制发行版文件拷贝到该目录下，即安装完毕。这个 Ant 目录就称为 ANT_HOME。

自己构建 Ant 比较麻烦一点，需要编译源代码。首先把解压后源代码放到 ANT_HOME 目录下，如果使用的是 Linux 系统，那么在 ANT_HOME 目录下运行“build.sh install”命令（不安装 docs 目录用“build.sh install-lite”命令）；如果是 Windows 操作系统，先进入 DOS 界面，在 ANT_HOME 目录下运行“build install”命令（不安装 docs 目录用“build install-lite”命令）。编译运行以后 ANT_HOME 目录下会多出几个文件夹，就是上面介绍的那几个。

6.2.2 配置 Ant 环境变量

在运行 Ant 之前需要做如下配置工作：

- 将 ant/bin 目录加入 PATH 环境变量里；
- 添加设定 ANT_HOME，指向安装 Ant 的目录。

不同的操作系统设置环境变量的方法也不同，下面仅介绍两种目前比较流行的、用户都比较熟悉的操作系统 Windows 和 Linux 中设置 Ant 环境变量的方法。

1. 在 Windows 环境下设定环境变量

在 Windows 环境下有两种设定环境变量的方法：一种是在 DOS 界面下设定，一种是 Windows 界面下设定。假设 Aut 安装在 D:\ant 目录下，设定方法如下。

（1）DOS 界面下设定环境变量

- 设定 ANT_HOME 环境变量，输入命令："set ANT_HOME= d:\ant"。
- 追加 PATH 环境变量，输入命令："set PATH=%PATH%;%ANT_HOME%\bin"。

（2）在 Windows 界面下设置环境变量

选中"我的电脑"点右键选"属性"→"详细设定"→"环境变量"→"新建"。在新弹出的新建对话框中输入以下内容：

- 变量名——ANT_HOME；
- 变量值——D:\ant。

然后编辑 PATH 环境变量，选中 PATH 环境变量，单击"编辑"按钮，在变量值后面追加"%ANT_HOME%\bin"。

2. 在 Linux 环境下设定环境变量

以 root 用户登录，用"vi"编辑器修改 root 目录下的".bash_profile"属性文件，假设 Ant 安装在/usr/local/ant，在文件里最后加两句话：

- 追加 ANT_HOME 环境变量："export ANT_HOME=/usr/local/ant"。
- 修改 PATH 环境变量："export PATH=$PATH:$ANT_HOME/bin"。

注意：修改环境变量后需要重启机器或者更换用户才能生效。

3. 运行 Ant

验证 Ant 是否可以运行只需要在提示符后面输入"ant"命令，然后回车，如果没有出现"ant"是非法命令的提示，则说明安装成功了。

如果出现"Buildfile: build.xml does not exist!Build failed"这样的提示，如图 6-1 所示，是因为所在目录下没找到 Ant 运行文件 build.xml，没有关系，在下面的章节中将介绍怎样编写 build.xml。



图 6-1　Ant 安装成功画面

有一点需要注意：不要将 Ant 的 ant.jar 文件放到 JDK/JRE 的 lib/ext 目录下。Ant 是一个应用程序，而 lib/ext 目录是为了 JDK 扩展用的（如 JCE，JSSE 扩展）。而且通过扩展装入的类会有安全方面的限制。

6.2.3　Ant 的依赖库

Ant 支持一些可选的 task。但是一个可选的 task 一般需要额外的库支持才能工作，可选的 task 与 Ant 内置的 task 是分开单独打包的。可选 task 包可以从表 6-1 中给的网

址下载。下载的 jar 文件应该放到 Ant 安装目录的 lib 文件夹下，这样 Ant 就能自动装入。

表 6-1 Ant 的依赖库及其下载网址

Jar 名	下 载 网 址
An XSL transformer like Xalan or XSL	http://xml.apache.org/xalan-j/index.html
jakarta-regexp-1.2.jar	http://jakarta.apache.org/regexp/
jakarta-oro-2.0.1.jar	http://jakarta.apache.org/oro/
junit.jar	http://www.junit.org
stylebook.jar	http://xml.apache.org
testlet.jar	http://java.apache.org/framework
antlr.jar	http://www.antlr.org
bsf.jar	http://oss.software.ibm.com/developerworks/projects/bsf
netrexx.jar	http://www2.hursley.ibm.com/netrexx
rhino.jar	http://www.mozilla.org
jpython.jar	http://www.jpython.org
netcompenents.jar	http://www.savarese.org/oro/downloads

6.3 运行 Ant

运行 Ant 非常简单，只要安装正确，在提示符下输入“ant”就可以了。如果没有输入任何参数时，Ant 会在当前目录下寻找 build.xml 文件。使用“-find”参数，Ant 会在上一级目录中寻找 buildfile，直至到达文件目录的根，如果想要让 Ant 使用其他的 buildfile，可以用参数“-buildfile file”，file 指的是你想要用的 xml 文件名。

用户还可以自己设定一些属性，以覆盖 buildfile 中指定的属性值。可以用“-Dproperty＝value”参数，这里 property 是用户指定的属性的名称，而 value 则是指属性的值。还可以用这种办法来指定环境变量的值，也可以用 property task 来存取环境变量，只要将“-DMYVAR=%MYVAR%”（Windows 下）或“-DMYVAR=$MYVAR”（Linux 下)传递给 Ant，用户就可以在自己的 buildfile 中用${MYVAR}来存取这些环境变量。

Ant 还有很多参数命令，输入“ant -help”命令，可以显示出所有命令。表 6-2 是 Ant 命令行选项的所有命令。

表 6-2 Ant 的参数命令及其功能说明

参数命令名称	功 能 说 明
-help	打印帮助信息

（续表）

参数命令名称	功　能　说　明
-projecthelp	打印工程帮助信息
-version	打印 Ant 版本信息
-diagnostics	打印有帮助的问题诊断和报告
-quiet, -q	执行 Ant 缺省任务
-verbose, -v	详细说明
-debug	打印调试信息
-emacs	生成 Log 信息
-logfile <file>	使用给定 LOG 文件
-l <file>	（同上）
-logger <classname>	执行 LOG 类
-listener <classname>	附加一个类的实例作为项目监听
-buildfile <file>	使用给定的 Build 文件
-file <file>	（同上）
-f <file>	（同上）
-D<property>=<value>	使用给定的属性值
-propertyfile <name>	用“-D”来加载所有具有优先权的属性
-inputhandler <class>	处理输入请求的类
-find <file>	从文件系统的根目录搜索 build 文件，并使用它

下面举几个运行命令的例子。

（1）ant

使用当前目录下的 build.xml 文件运行 Ant，执行缺省的 target。

（2）ant –buildfile test.xml

使用当前目录下的 test.xml 运行 Ant，执行缺省的 target。

（3）ant –buildfile test.xml main

使用当前目录下的 test.xml 运行 Ant，执行叫做 main 的 target。

（4）ant –buildfile test.xml –Dbuild=build/classes main

使用当前目录下的 test.xml 运行 Ant，执行一个叫 main 的 target，并设定 buiid 属性的值为 build /class。

6.4　build.xml 语法详解

Ant 的 buildfile 是用 XML 写的，每个 buildfile 含有一个 project。buildfile 中每一个 task 元素有一个 id 属性，可以用这个 id 值引用指定的任务。这个值必须是惟一的。

6.4.1 项目（project）

project 是 buildfile 文件的根元素，它有三个属性（见表 6-3）。

表 6-3 project 属性

属性名	内 容
name	项目名称
Default	表示默认的 target，运行 ant 时如果不指定 target，则用 default 指定的 target
Basedir	设置为 "." 表示 build.xml 所在的路径

项目的描述是以一个顶级的<description>元素的形式出现。一个项目可以定义一个或多个 target。一个 target 是一系列要执行的。执行 Ant 时，可以选择要执行的 target。如果没有指定 target 时，默认使用 project 的 default 属性所确定的 target。

6.4.2 目标（target）

一个 target 可以依赖于其他的 target。例如，可能会有一个 target 用于编译程序，一个 target 用于执行程序。在执行程序之前必须先编译通过，所以执行程序的 target 依赖于编译 target。Ant 会处理这种依赖关系。然而，应当注意到，Ant 的 depends 属性只制定了 target 应该被执行的顺序。如果被依赖的 target 无法执行，这种 depends 对于指定了依赖关系的 target 就没有影响。Ant 会依照 depends 属性中 target 出现的顺序（从左到右）依次执行每个 target。要记住的是，只要某个 target 依赖于一个 target，后者就会被先执行。例如：

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="C"/>
```

如果要执行 target D，从它的依赖属性来看，可能认为先执行 C，然后执行 B，最后 A 被执行。其实错了，C 依赖于 B，B 依赖于 A，所以先执行 A，然后执行 B 和 C，最后执行 D。

一个 target 只能被执行一次，即使有多个 target 依赖于它。如果某些属性被设定，或者没被设定，才执行某个 target。这样，允许根据系统的状态来更好地控制 build 的过程。如果想要让一个 target 这样做，就应该在 target 元素中加入 if 或者 unless 属性，带上 target 应该有所判断的属性。例如：

```
<target name="build-module-A" if="module-A-present"/> 或
<target name="build-own-fake-module-A" unless="module-A-present"/>
```

如果没有 if 或 unless 属性，target 总会被执行。可选的 description 属性可以用来提供关于 target 的一行描述，这些描述可由-projecthelp 命令行选项输出。

target 的各种属性见表 6-4。

表 6-4　target 属性

属 性 名	内　　容
name	target 的名字
depends	用逗号分隔的 target 的名字列表，也叫做依赖表
if	执行 target 所需要设定的属性名
Unless	执行 target 需要清除设定的属性名
Description	关于 target 功能的简短描述

建议的顶级目标的名称及内容如表 6-5 所示。

表 6-5　顶级目标名

目标的名字	内　　容
Build	构建应用系统
Test	运行 JUnit 测试
Clean	清空输出目录
Deploy	将 jar 或 war 等文件装载到执行系统中
Publish	输出源代码和二进制文件到发布站点
Fetch	从 CVS 树得到最新源代码
docs/javadocs	输出文档文件
All	执行 clean, fetch, build,test, dos/javadocs, deploy
Main	默认的构建过程

内建目标的名称及内容如表 6-6 所示。

表 6-6　内建目标名

内建目标的名称	内　　容
Init	初始化特性和执行其他初始化任务，读取个性化特性文件
Init-debug	初始化调试特性
Init-release	初始化发布特性
Compile	执行实际编译
link/jar	建立 jar 文件或等价文件
Staging	在将应用程序转移到产品站点前执行预部署过程及测试

简单项目的开发目录结构如表 6-7 所示。

表 6-7 项目开发目录结构

目 录 名	内 容
bin	存放普通的二进制文件和脚本文件
build	用以存放构建结果的目录树，由 ant 创建且可以被 "clean" 目标清空
dist	用以存放可发布的开发结果，由 ant 创建且可以被 "clean" 目标清空
doc	手写文档
lib	存放在构建应用系统时需要装入的 Java 库
src	存放 Java 源码，目录的层次结构要符合包命名的要求

6.4.3 任务（task）

每一个 task 都是一段可以执行的代码。一个 task 可以有多个属性（也有的可以称为变量）。属性只可能包含对 property 的引用。这些引用会在 task 执行前被解析。

下面的这句话是 task 的基本构造形式：

```
<name test1="car1"
```

其中：name 是 task 的名字，test1 是属性名，car1 是属性值。

Ant 有一套内置的（built-in）task，以及一些可选的 task，但用户也可以编写自己的 task。所有的 task 都有一个 task 名字属性。Ant 用属性值来产生日志信息。

可以为 task 命名一个 id 属性：

```
<taskname id="taskID"…/>
```

这里，taskname 是 task 的名字，taskID 是这个 task 的惟一标识符。通过这个标识符，可以在脚本中引用相应的 task。例如，可以在脚本中这样写：

```
<script ... task1.setBo("car");></script>
```

设定某个 task 实例的 bo 属性，在另一个 task 中（用 Java 编写），可以利用下面的语句存取相应的实例。

```
project.getReference("task1")
```

有两点需要注意：

（1）如果 task1 还没有运行，就不会被生效（例如不设定属性），如果在随后配置它，所作的一切都会被覆盖。

（2）未来的 Ant 版本可能不会兼容这里所提到的属性，因为很有可能根本没有 task 实例，只有 proxies。

6.4.4 属性（property）

一个 project 可以有很多的 property。可以在 buildfile 中用 property task 来设定，或

在 Ant 之外设定。一个 property 有一个名字和一个值。property 可用于 task 的属性值。这是通过将属性名放在“${”和“}”之间并放在属性值的位置来实现的。例如，如果有一个 property builddir 的值是“WEB-INF”，这个 property 就可用于属性值：${builddir}/classes，这个值就可被解析为 WEB-INF/classes。

1. Ant 内置属性

如果使用了<property> task 定义了所有的系统属性，Ant 允许使用这些属性。例如，${os.name}对应操作系统的名字。要想得到系统属性的列表，可参考 The Javadoc of System.getProperties。除了 Java 的系统属性，Ant 还定义了一些自己的内置属性，如表 6-8 所示。

表 6-8　Ant 内置属性

属 性 名	内　　容
basedir	project 基目录的绝对路径（与<project>的 basedir 属性一样）
ant.file	buildfile 的绝对路径
ant.version	Ant 的版本
ant.project.name	当前执行的 project 的名字，由<project>的 name 属性设定
ant.java.version	Ant 检测到的 JVM 的版本，目前的值有 1.1、1.2、1.3 和 1.4

下面举例做具体说明一下，例 6-1 综合了上面介绍的各种语法，主要实现了编译、部署、打包.jar 文件，以及清除自动生成的文件夹。

例 6-1：

```
<project name="Test" default="dist" basedir=".">
    <!--设置属性值 -->
    <property name="src" value="."/>
    <property name="build" value="build"/>
    <property name="dist" value="dist"/>
    <target name="init">
      <!--创建用于编译的 build 文件夹 -->
      <mkdir dir="${build}"/>
    </target>
    <target name="compile" depends="init">
      <!--编译 src 目录下的 java 代码，将生成的 class 文件放到 build 文件夹里 -->
      <javac srcdir="${src}" destdir="${build}"/>
    </target>
    <target name="dist" depends="compile">
      <!--创建存放 jar 包的 lib 文件夹 -->
      <mkdir dir="${dist}/lib"/>
      <!--将 build 文件夹下的所有东西打包成 jar 文件放到 dist 文件夹下的 lib 文件夹里-->
      <jar jarfile="${dist}/lib/Test-${DSTAMP}.jar" baseDir="${build}"/>
    </target>
    <target name="clean">
      <!--清除 build 和 dist 文件夹 -->
      <delete dir="${build}"/>
      <delete dir="${dist}"/>
    </target>
</project>
```

6.4.5 过滤器（token filters）

一个 project 可以有很多标记，这些标记在文件拷贝时会被自动扩展，这要求在支持这一行为的任务中选择过滤拷贝功能。这一功能可用过滤器任务在 buildfile 中设定。

既然这很可能是一个有危害的行为，文件中的标记必须采取“@标记@”的形式，这里的“标记”是过滤器任务中设定的标记名。这种标记语法与其他 build 系统执行类似过滤的语法相同，而且与大多数的编程和脚本语言以及文档系统并不冲突。

注意：如果在一个文件中发现了一个“@标记@”形式的标记，但没有过滤器与这个标记关连，则不会发生任何事。因此，没有转义方法，只要用户为标记选择合适的名字，就不会产生问题。

警告：如果用户在拷贝二进制文件时打开过滤功能，用户文件有可能被破坏。这个功能只针对文本文件。

6.4.6 路径指定元素

可以用“:”和“;”作为分隔符，指定类似 PATH 和 CLASSPATH 的引用。Ant 会把分隔符转换为当前系统所用的分隔符。

1. 通过<classpath>元素指定路径

当需要指定类似路径的值时，可以使用嵌套元素。一般的形式是：

```
<classpath>
<pathelement path="${classpath}"/>
<pathelement location="lib/helper.jar"/>
</classpath>
```

这里，location 属性指定了相对于 project 基目录的一个文件和目录，而 path 属性接受逗号或分号分隔的一个位置列表。path 属性一般用作预定义的路径，其他情况下，应该用多个 location 属性。

为简单方便，classpath 标签支持自己的 path 和 location 属性。上面的代码可以简写成：

```
<classpath path="${classpath}" location="lib/helper.jar"/>
```

2. 通过<fileset>元素指定路径

也可通过<fileset>元素指定路径。构成一个 fileset 的多个文件加入。以“**/*”的形式将需要的多个文件导入，顺序是未定的。

<fileset>元素在 build.xml 中用得非常广泛，有些可选的任务需要额外的库支持才能运行，<fileset>元素一般用于导入这些可选的任务需要的 jar 文件。下面是<fileset>元素的一般形式：

```
<classpath>
<pathelement path="${classpath}"/>
<fileset dir="lib">
<include name="**/*.jar"/>
```

```
</fileset>
<pathelement location="classes"/>
</classpath>
```

上面的例子构造了一个路径值包括：${classpath}的路径，跟着 lib 目录下的所有 jar 文件，接着是 classes 目录。

3. 通过< path >元素指定路径

如果想在多个任务中使用相同的路径结构，可以用<path>元素定义它们（与 target 同级），然后通过 id 属性引用。

一个路径结构可能包括对另一个路经结构的引用（通过嵌套<path>元素实现）：

```
<path id="base.path">
<pathelement path="${classpath}"/>
<fileset dir="lib">
<include name="**/*.jar"/>
</fileset>
<pathelement location="classes"/>
</path>
<path id="tests.path">
<path refid="base.path"/>
<pathelement location="testclasses"/>
</path>
```

前面所提的关于<classpath>的简洁写法对于<path>也是可以的，如：

```
<path id="tests.path">
<path refid="base.path"/>
<pathelement location="testclasses"/>
</path>
```

可写成：

```
<path id="base.path" path="${classpath}"/>
```

6.4.7　命令行变量

有些 task 可接受参数，并将其传递给另一个进程。为了能在变量中包含空格字符，可使用嵌套的<arg/>元素。<arg/>元素拥有的属性如表 6-8 所示。

表 6-9　arg 元素属性

属性名	内　　容
value	一个命令行变量；可包含空格字符，只能用一个
line	空格分隔的命令行变量列表
file	作为命令行变量的文件名；会被文件的绝对名替代
path	作为单个命令行变量的路径字符串；或作为分隔符，Ant 会将其转变为特定平台的分隔符

例如：

<arg value="-l -a"/> 表示一个含有空格的单个的命令行变量。

<arg line="-l -a"/> 表示两个空格分隔的命令行变量。

<arg path="/dir1;/dir2:/dir3"/> 是一个命令行变量，其值在 DOS 系统上为\dir1;\dir2;\dir3；在 Linux 系统上为/dir1:/dir2:/dir3。

buildfile 元素的 id 属性可用来引用这些元素。如果多个任务用到相同的 CLASSPATH 路径，用户不需要一遍遍地复制相同的 XML 代码块，这一属性就可以实现此目的。

下面是多次使用<classpath>结构情况：

```
<project ... >
    <target ... >
        <jar ...>
            <classpath>
                <pathelement location="lib/"/>
                <pathelement path="${java.class.path}/"/>
                <pathelement path="${additional.path}"/>
            </classpath>
        <jar .../>
    </target>
<target ... >
      <javac ...>
          <classpath>
                <pathelement location="lib/"/>
                <pathelement path="${java.class.path}/"/>
                <pathelement path="${additional.path}"/>
          </classpath>
       </javac>
    </target>
</project>
```

它可以写成下面的形式：

```
<project ... >
    <path id="project.class.path">
        <pathelement location="lib/"/>
        <pathelement path="${java.class.path}/"/>
        <pathelement path="${additional.path}"/>
    </path>
    <target ... >
        <jar ...>
            <classpath refid="project.class.path"/>
        </jar>
    </target>
    <target ... >
        <javac ...>
            <classpath refid="project.class.path"/>
        </javac>
    </target>
</project>
```

请参看 Ant 自带的文档或者登录 Ant 官方网站（http://ant.apache.org/）以了解更多、更新的关于 Ant 方面的信息。

6.5　使用 Ant 插件

Eclipse 平台很好的集成了 Ant，允许从插件运行 Ant 构建文件，并使用扩展点添加新的 Ant 任务和类型。本节将通过一个实例介绍如何使用 Ant 插件。

6.5.1　创建 Ant 构建文件

下面创建一个 Ant 构建文件，并且试着用 Eclipse Ant 编辑器编辑运行。所谓的 Ant 构建文件就是文本文件。因此，在 Eclipse 中创建 Ant 构建文件是非常简单的，具体步骤如下。

（1）右击项目名→“新建”→“文件”，如图 6-2 所示。



图 6-2　新建文件

（2）输入或选择父文件夹，然后输入文件名，文件名以“.xml”为扩展名，如“Hello.xml”。单击“完成”，Ant 构建文件即创建完成，如图 6-3 所示。



图 6-3 创建 xml 文件

6.5.2 编辑 Ant 构建文件

因为 Ant 构建文件是简单的文本文件，所以可以使用任何文本编辑器来编辑它们。但是使用 Eclipse Ant 编辑器有几个优点（包括语法着色、内容辅助和大纲视图）。为使读者熟悉 Eclipse Ant 编辑器，现使用该编辑器编辑一个简单的 Ant 构建文件。

1. 添加 Eclipse Ant 编辑器

选择菜单栏“窗口”→“首选项”→“工作台”→“文件关联”→“*.xml”文件类型，如图 6-4 所示。如果列表中没有这个文件类型的话，需要事先点“添加（A）”添加一个“*.xml”文件类型。

按“添加（D）”按钮会弹出一个编辑器选择的窗口，选中“Ant 编辑器”，单击“确定”即可，如图 6-5 所示。

至此，就成功地完成了 Eclipse Ant 编辑器的添加，点“确定”完成，如图 6-6 所示。



图 6-4　添加文件类型

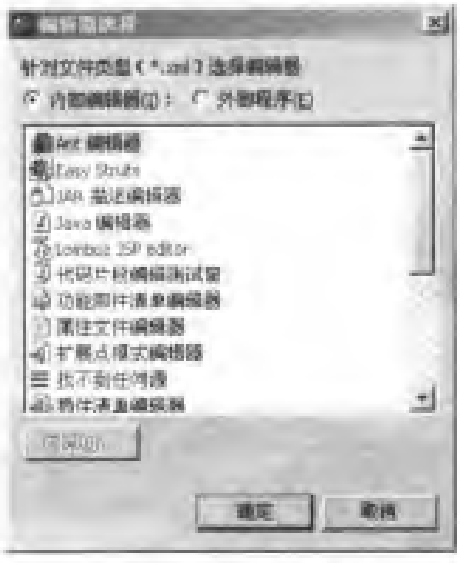


图 6-5　选择编辑器

2. 使用 Ant 编辑器编辑 Hello.xml 文件

右击 Hello.xml 文件，选择“打开放式”下拉菜单，可以看到刚才添加的 Ant 编辑器，选中它。现在就可以用 Ant 编辑器编辑 Hello.xml 文件了，如图 6-7 所示。



图 6-6　添加完毕画面

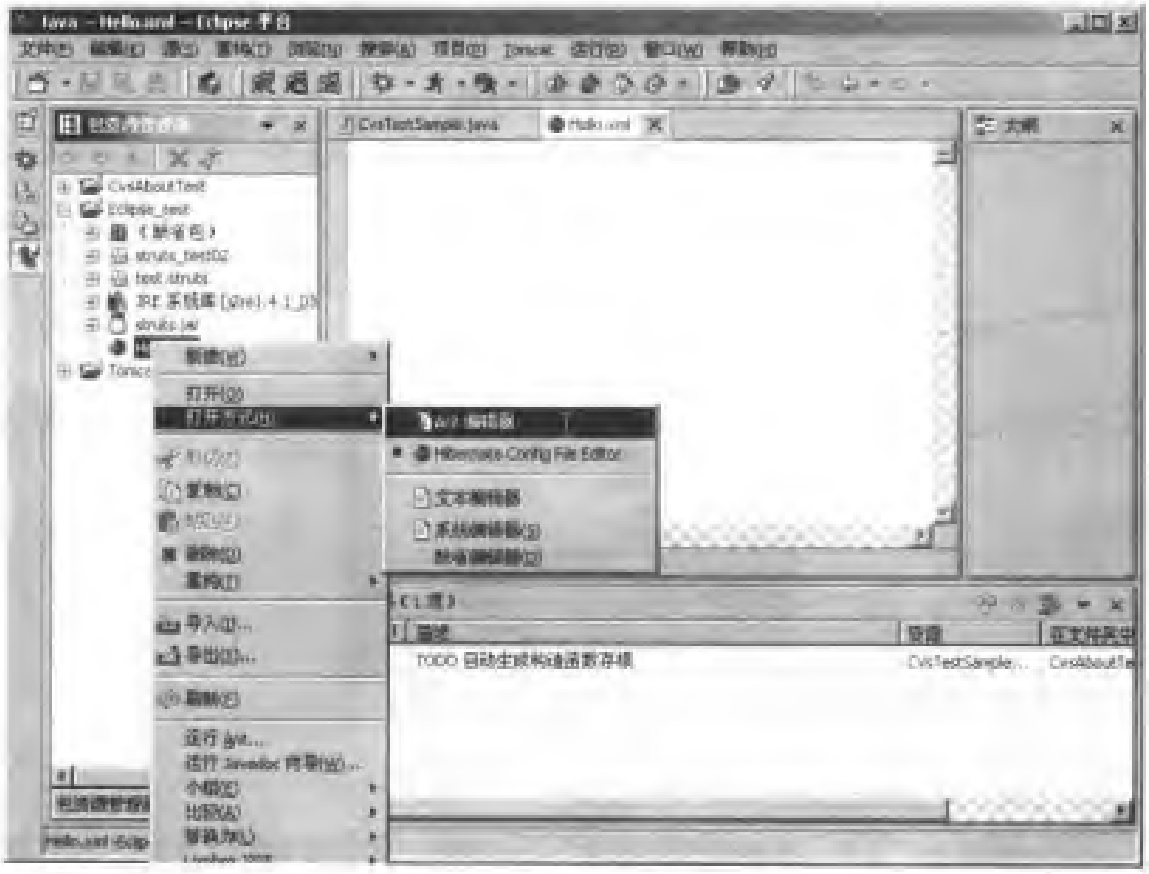


图 6-7　用 Ant 编辑器编辑 xml 文件

在 Ant 编辑器中输入以下内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="Hello World" default ="Hello" basedir=".">
        <property name ="Hello Text" value="Hello"/>
        <target  name="Hello">
                <echo>${Hello Text}</echo>
        </target>
</project>
```

在 Ant 编辑器中可以看到 Hello.xml 文件的高亮显示，如图 6-8 所示。

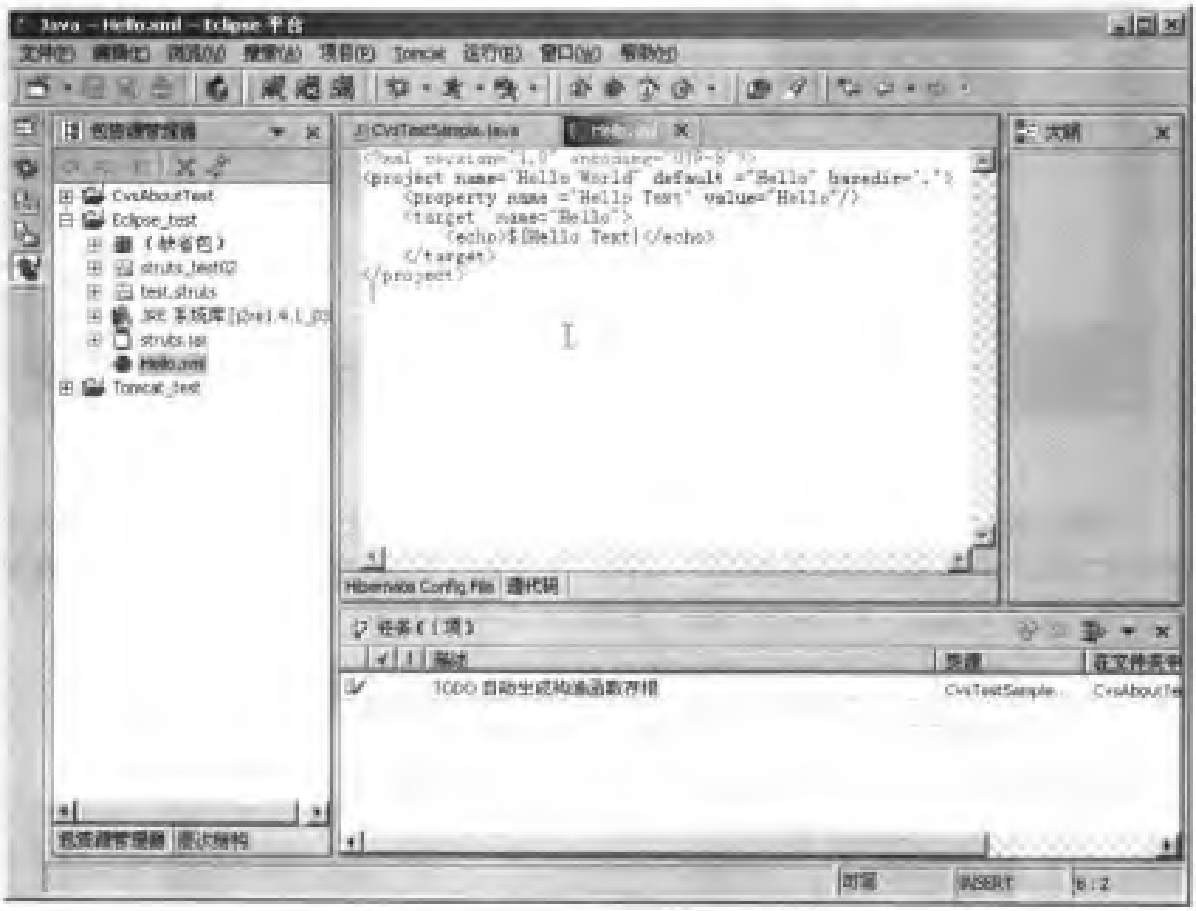


图 6-8　用 Ant 编辑器编辑的.xml 文件

6.5.3　运行 Ant 构建文件

带有.xml 扩展名的任何文件都可作为 Ant 构建文件运行。当然，并非所有这种文件都是真正的 Ant 构建文件，但是，如果错误地将非 Ant .xml 文件作为 Ant 构建文件运行，也不会有产生什么危害。下面运行刚编辑的 Hello.xml 文件。右击 Hello.xml 文件，选择“运行 Ant...”，如图 6-9 所示。



图 6-9　运行 Ant

在弹出的对话框上选择要执行的目标。此对话框允许配置 Ant 构建文件的运行方式的许多方面，但是现在我们将着重于“目标”选项卡，它允许选择要运行的 Ant 目标及其次序。选择完成后，按“运行（U）”按钮，如图 6-10 所示。

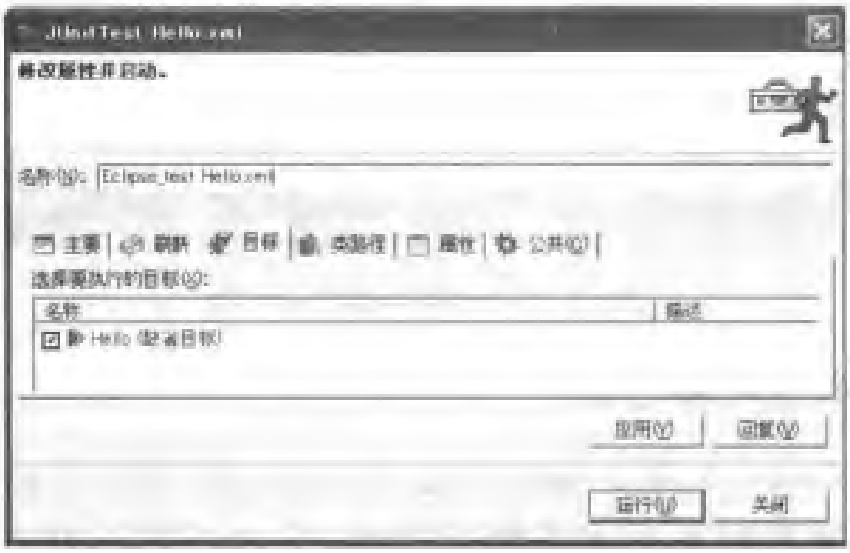


图 6-10　修改属性并启动

Ant 构建文件在运行过程中，同时将输出信息发送至“控制台”视图。运行结果如图 6-11 所示。

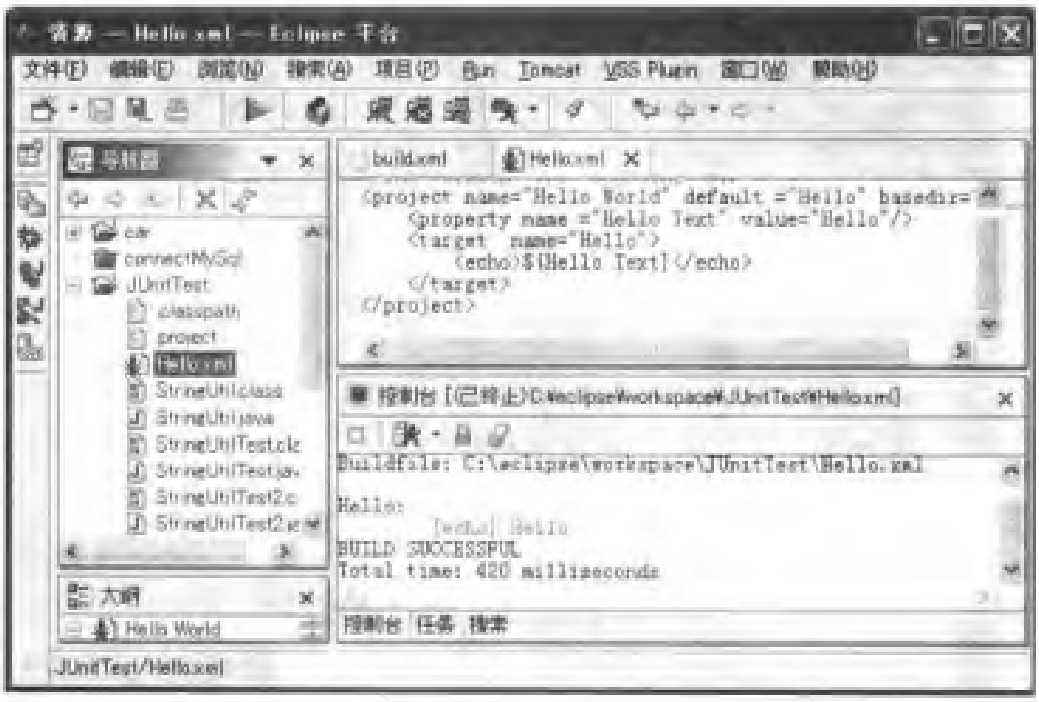


图 6-11　运行结果

6.5.4　保存和重用 Ant 选项

当在上一步中运行 Ant 构建文件 Hello.xml 时，将目标的选项以及“运行 Ant”对话框中的所有其它选项都保存在称为“启动配置”的实体中。“启动配置”中包含按特定方式运行单个 Ant 构建文件所必需的所有详细信息，与单个 Ant 构建文件相关联的多个启动配置是完全有效的。因此，除了在上一步骤中创建的启动配置之外，还可以为同一个构建文件创建第二个启动配置（指定相同的目标但是按相反的次序）。启动配置真正的好处在于：现在通过简单地指定相应的启动配置就可以在任一配置中快速运行 Ant 构建文件。

下面为 Hello.xml 创建两个启动配置。

（1）打开“运行”菜单，选中“外部工具”下的“外部工具”，如图 6-12 所示。

（2）“外部工具”对话框弹出，该对话框与前面看到的“运行 Ant”对话框很相似。事实上，除了在“外部工具”对话框中可以选择想要查看和编辑哪个启动配置之外，其他是完全一样的。如果在左边的树中选择运行构建文件时创建的启动配置（Tomcat_testHello.xml），则右边的选项卡中显示出此启动配置的选项，如图 6-13 所示。

（3）将启动配置文件名称改为“Tomcat_testHelloWorld.xml”，单击“应用”。然后在左边的树中，选中构建文件，点右键，复制。创建 HelloWorld 构建文件的启动配置的副本。然后选中副本，将新的配置重命名为“Tomcat_testWorld Hello.xml”。单击“应用”，然后“运行”，如图 6-14 所示。

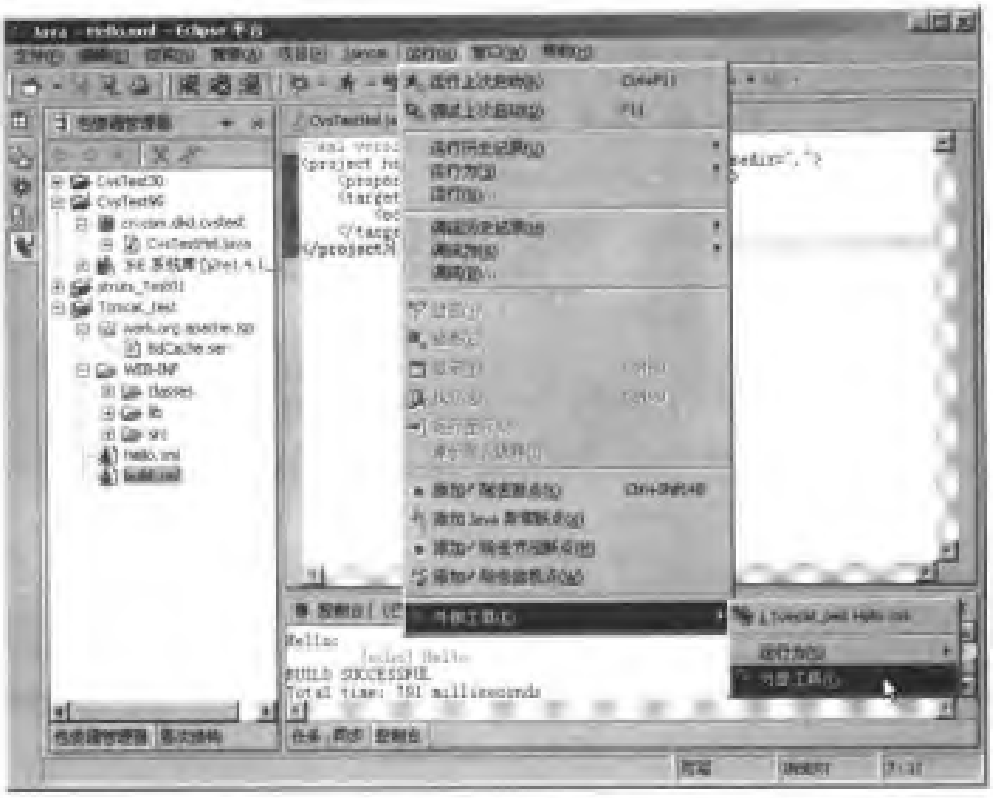


图 6-12　添加外部工具



图 6-13　外部工具创建、管理和运行配置的画面



图 6-14　修改后的外部工具窗口画面

此时，Hello.xml 用第二个启动配置文件运行。

（4）使用 Ant 视图。Eclipse 提供使用 Ant 构建文件的标准视图（即 Ant 视图）。Ant 视图为树型结构，将 Ant 构建文件作为顶级条目显示，而目标和子目标则作为子代显示。此视图的主要优点在于：与在“导航器”中搜索所有 Ant 构建文件相比，可以在一个位置使用它们。

（5）显示 Ant 视图：

① 打开“窗口”菜单，选择“显示视图”，选中“Ant”，如图 6-15 所示。

② Eclipse 平台将显示 Ant 视图，如图 6-16 所示。

6.6　用 build.xml 构建 Ant 部署文件

build.xml 文件是带有许多对部署插件有用的目标的缺省 Ant 构建文件。可以为它创建 build.xml 文件来管理插件部署。

1．新建一个 build.xml 部署文件

首先新建一个名为“AntTest”的工程，然后做一个简单的 Java 程序“HelloWorld.java”，输出“HelloWorld”就可以了（具体做法，请参考第 2 章）。接着，新建一个 build.xml 部署文件，其内容如例 6-2 所示。



图 6-15　显示 Ant 视图



图 6-16　Ant 视图

例 6-2：build.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project name = "build" default="makejar" basedir=".">
        <target name="main" >
            <--在根目录下编译 ".java" 文件-->
                <javac srcdir="." destdir="."/>
            <!--运行编译后的 ".class" 文件-->
                <java classname="HelloWorld">
                    <classpath>
                        <pathelement path="."/>
                    </classpath>
                </java>
```

```
    </target>
        <target name="makejar" depends="main"
                description="Create a jar for the AntTest project">
            <!--将根目录下的文件打包成 jar -->
            <jar jarfile="HelloWorld.jar" includes="*.class" basedir="."/>
        </target>
    </project>
```

2. 运行 build.xml 文件

文件保存后，选中导航图中的“build.xml”，右击“运行 Ant...”。运行结果可能会有如图 6-17 所示的错误。这个错误是由于没有添加 JDK 的 jar 文件而引起的。

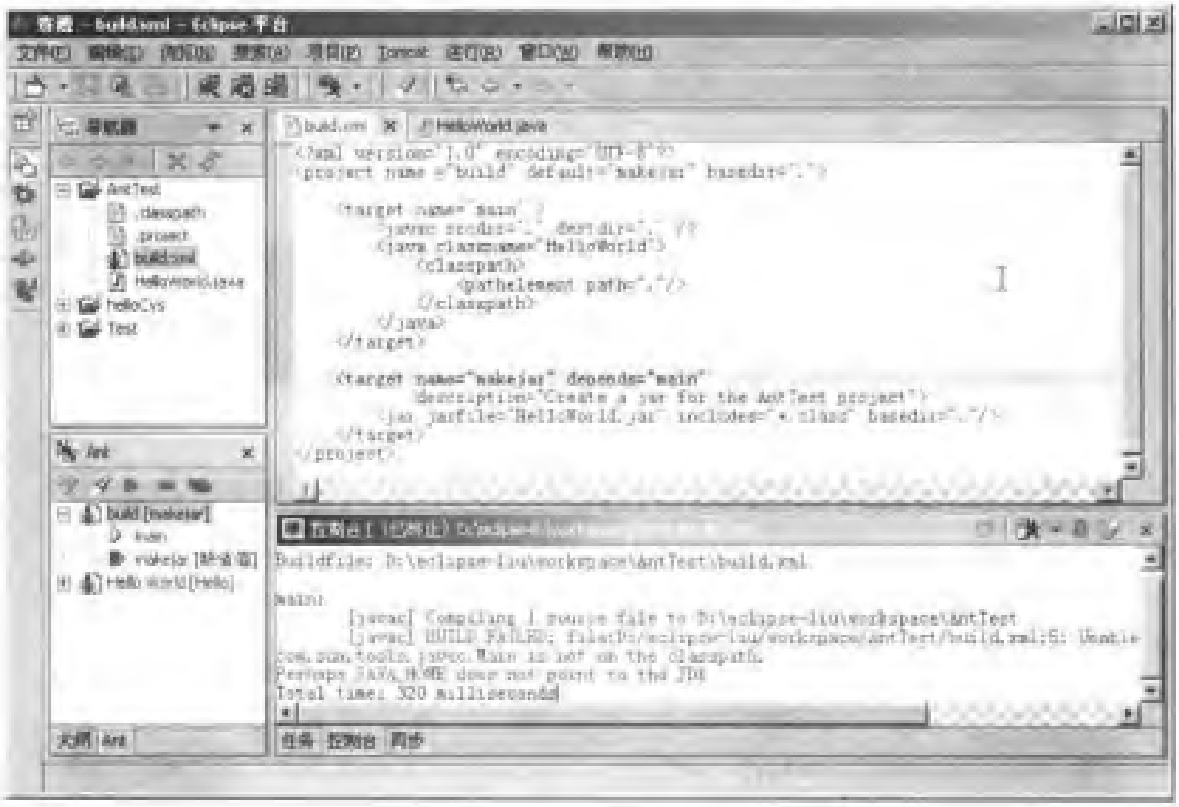


图 6-17　运行结果画面

重新选中导航图中的“build.xml”，右击选择“运行 Ant...”。这时弹出一个“修改属性并启动”对话框，如图 6-18 所示。选择“类路径”选项卡。

单击“添加 Jar”按钮，找到 JDK 所在的文件夹下，将 lib 文件夹下的所有 jar 文件添加进去，如图 6-19 所示。

单击“运行”按钮，运行的结果显示在控制台上面，如图 6-20 所示。

打印的语句“HelloWorld”显示在控制台上，而且在工作目录下编译生成了“HelloWorld.class”和“HelloWorld.jar”文件。

选中导航器里的项目名“AntTest”，点右键，选择“刷新”后，就可以看到多了两个文件，它们就是“HelloWorld.class”和“HelloWorld.jar”文件，如图 6-21 所示。



图 6-18　修改属性并启动画面

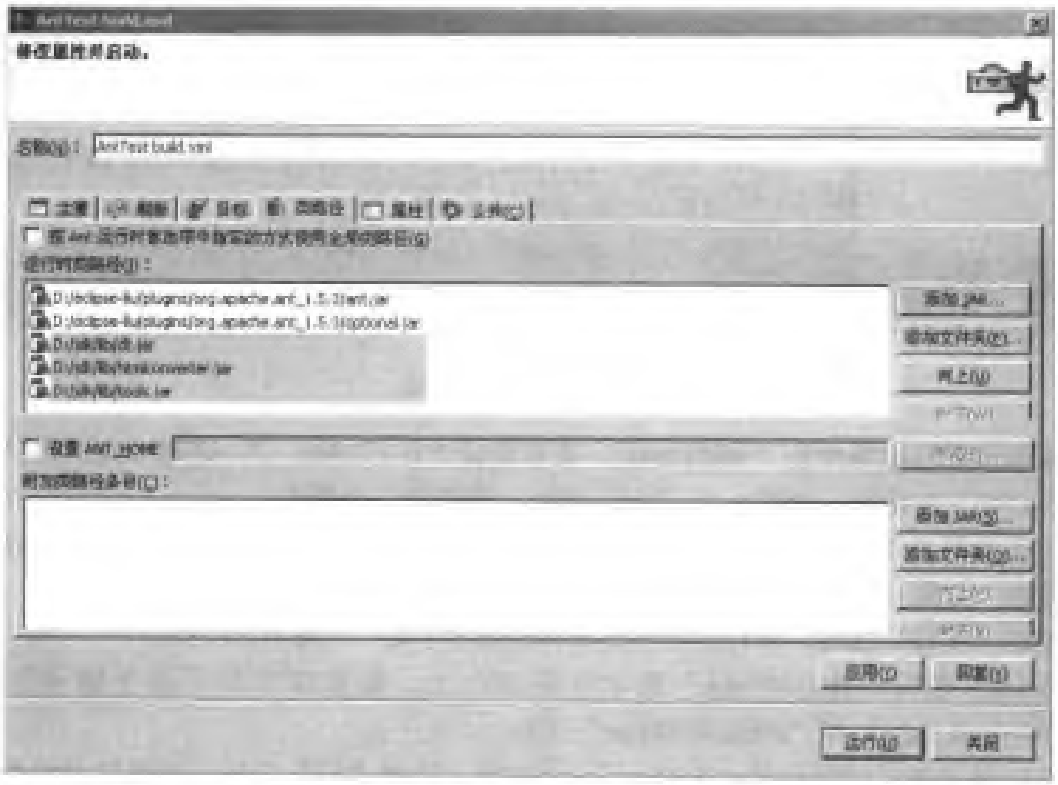


图 6-19　添加 JDK 的 jar 文件



图 6-20　运行结果画面



图 6-21　自动生成编译文件和打包文件

6.7 Ant 和 JUnit 结合进行单元测试

JUnit 和 Ant 的结合犹如马和马车的关系，Ant 将构建及部署的程序自动化，JUnit 则将测试自动化。将二者整合在一起，Ant 便能够将构建部署及测试的程序自动化。利用 Ant 的 JUnit 任务和其子任务 test 可以在 Ant 配置文件中执行单元测试。

下面以具体的例子，感受一下自动化构建部署与测试一体带来的便利。

1. 导入 junit.jar 文件

首先，为了让 JUnit 和 Ant 结合在一起，需要下载 junit.jar 文件，junit.jar 的下载网址为 http://unc.dl.sourceforge.net/junit/。将下载的 JUnit 压缩文件解压缩后，把 junit.jar 拷贝到 ANT_HOME/lib 文件夹下。

2. 新建项目和 Java 类

用 Eclipse 新建一个项目“car_test”，在项目根目录下新建 etc\xsl 文件夹，从 %ANT_HOME%\etc 下拷贝 junit-noframes.xsl 文件至 D:\car\etc\xsl 子目录下。在 src 目录下新建 Car.java 文件，其内容见例 6-3。

例 6-3：Car.java

```
public class Car {
    public int getNumber() {
        return 6;
    }
}
```

3. 新建测试类

Car.java 文件保存后，在资源图上选中该文件，点右键，选择“新建”→“测试类”（具体做法参考第 5 章“Eclipse 与 JUnit”）。测试类 CarTest.java 文件内容如例 6-4 所示。

例 6-4：CarTest.java

```
import junit.framework.*;
    public class testCar extends TestCase {
    protected int expectedWheels;
    protected Car myCar;
    public testCar(String name) {
        super(name);
    }
    protected void setUp() {
        expectedWheels = 4;
        myCar = new Car();
    }
    public static Test suite() {
        return new TestSuite(testCar.class);
    }
    public void testGetWheels() {
        assertEquals(expectedWheels, myCar.getWheels());
    }
}
```

4. 新建 Ant 执行文件 build.xml 和属性文件 build.properties

（1）build.properties 文件里定义一些 Path 变量，其内容如例 6-5 所示。

例 6-5：build.properties

```
src.dir=src
build.dir=classes
report.dir=report
jar.dir=war
xsl.dir=etc\xsl
```

（2）build.xml 文件内容如例 6-6 所示。

例 6-6：build.xml

```
<?xml version="1.0"?>
<project default="build" basedir=".">
<!--导入属性文件-->
<property file="build.properties" description="Transfer property
from .properties file"/>
    <!--创建工作目录-->
    <target name="build" description="Build the work directory">
        <mkdir dir="${build.dir}" />
        <mkdir dir="${report.dir}" />
        <!--编译 src.dir 下的 java 文件到 build.dir -->
        <javac srcdir="${src.dir}" destdir="${build.dir}">
            <include name="**/*.java" />
        </javac>
    </target>
    <!--用 junit 任务测试-->
    <target name="junit" depends="build"  description="Execute Unit Tests">
        <tstamp/>
        <junit printsummary="yes">
                <classpath>
                        <pathelement location="${build.dir}"/>
                </classpath>
                <formatter type="xml" />
                <batchtest todir="${report.dir}">
                        <fileset dir="${src.dir}">
                                <include name="**/testCar.java" />
                        </fileset>
                        </batchtest>
        </junit>
        <!--生成测试报告-->
        <junitreport todir="${report.dir}">
                <fileset dir="${report.dir}">
                        <include name="TEST-*.xml"/>
                </fileset>
                <report format="noframes" styledir="${xsl.dir}"
            todir="${report.dir}"/>
        </junitreport>
    </target>
    <!--打包 jar 文件-->
    <target name="jar" depends="build" description=" Create a jar ">
        <jar jarfile="${jar.dir}/car_test.jar" basedir="${build.dir}"/>
    </target>
    <!--清除文件夹-->
    <target name="clean" description="Delete the work directory">
        <delete includeEmptyDirs="true">
```

```
            <fileset dir="${build.dir}"/>
            <fileset dir="${report.dir}"/>
        </delete>
    </target>
</project>
```

5. 运行 Ant

执行 build.xml 中的不同 target，以分别完成编译、测试、生成测试报告、部署、打包。

本章小结

本章详细介绍了有关 Ant 的知识并讲解了使用 Ant 插件创建 Ant 构建文件部署管理项目，以及结合 JUnit 进行单元测试。Ant 部署项目灵活、强大，可以自动完成许多工作。如果要深入了解 Ant，可以到 Ant 官方网站（http://ant.apache.org/）上查阅相关文档资料。

第 7 章　使用 Eclipse 协同 CVS 进行版本管理

本章介绍如何使用 CVS 协同 Eclipse 进行版本控制。版本控制就是在软件开发过程中，对软件源代码文件的修改进行记录、保存的过程。版本控制对于团队开发是重要的，对于个人开发者也是项目复杂化时不可缺少的工具。本章包含如下内容：

- 什么是 CVS
- CVS 服务器的安装
- 在 Linux 平台上的配置 CVS 服务器
- 在 Windows NT 平台上的配置 CVS 服务器
- 应用 CVS 系统协助开发的一个简单的流程
- CVS 的几个基本概念
- Eclipse 和 CVS 的连接
- 启动一个新项目
- 应用 Eclipse 从 CVS 资源库检出代码
- 在开发过程中 Eclipse 协同 CVS 开发的基本操作

7.1　什么是 CVS

CVS 是在 UNIX 体系里的 SCCS 和 RCS 的基础上开发的，用于源代码版本管理的产品。它是客户机 / 服务器结构的产品，可用于多种平台（Linux，Windows），目前已经发展成为软件开发人员在开发过程中进行版本管理的主流产品之一。

CVS 的主要功能是在 CVS 服务器端记录软件开发过程中源文件的历史。当同一组的开发人员对同一个项目进行开发时，CVS 会将他们进行区分。这样，每个开发人员可以在自己的客户端进行独立的开发工作，在开发人员向 CVS 服务器提交新的文件后，由 CVS 资源库（CVS repository）统一进行合并。

7.2　CVS 服务器的安装

在这一节里将配置 CVS 服务器，首先在 Linux 平台上安装 CVS 服务器，本例采用 Redhat Linux 9.0，主机名是 DKD-SV1。之后，在 Window NT 平台上安装 CVS 服务器，

用户名为 administrator。

1. 确认系统中是否安装了 CVS

以超级用户（root）身份登录系统，执行下面的指令：

```
[root@DKD-SV1 home] #rpm -qa|grep cvs
```

如果输出类似[root@DKD-SV1 home] #cvs-1.11.2-10 的信息，说明系统已经安装有 CVS，如果系统是 Linux，则有很大可能已经默认安装了 CVS 服务器软件包，否则就需要从安装光盘中安装 CVS 的 rpm 包，或者到网址 http://www.cvshome.org 下载最新版本的 CVS。如果你的系统已经安装了 CVS 请跳过这一节，在 7.3 节中配置 CVS 服务器。

CVS 是在 Unix 体系里的 SCCS 和 RCS 的基础上发展而来的，它在 Unix/Linux 平台上已经是一个很成熟的产品了。随着 CVS 在版本管理系统市场上的主流地位的确立，越来越多的人希望将 CVS 服务器移植到 Window 平台上。经过开发者的努力，Windows NT/2000 版的 CVS 服务器终于推出了。在这里，编者建议使用 Unix/Linux 平台安装 CVS。

2. 在 Unix/Linux 平台安装 CVS 服务器

从 http://www.cvshome.org 下载最新版本的 CVS，本书下载的是 cvs-1.11.1p1.tar.gz，将安装文件保存到服务器上。以超级用户（root）身份登陆，并到存放文件的路径下（在本书的环境里是/root/cvs_install/），运行：

```
[root@DKD-SV1 cvs_install]# tar zxvf cvs-1.11.1p1.tar.gz
```

查看一下解压后的文件夹下的情况：

```
[root@DKD-SV1 cvs_install]# ls -a /root/cvs_install/
.  ..  cvs-1.11.1p1  cvs-1.11.1p1.tar.gz
```

之后，进入 cvs-1.11.1p1 文件夹，运行：

```
[root@DKD-SV1 cvs-1.11.1p1]# ./configure
[root@DKD-SV1 cvs-1.11.1p1]# make
[root@DKD-SV1 cvs-1.11.1p1]# make install
```

至此，如果系统没有给出错误信息，就说明 CVS 服务器已经安装完毕了。

3. 在 Windows NT/2000 平台安装 CVS 服务器

可以从 http://www.cvsnt.org/（该网站是为 Windows NT/2000 平台的 CVS 服务器的发布而建立的）下载最新版本的 CVSNT，在这里使用 CVSNT 2.1.1。

在服务器上双击安装图标后，进行安装。在安装过程中，会自动在系统属性中添加 CVS 的安装路径。

本书采用的 CVS 安装路径是：G:\Program Files\cvsnt。请在安装 CVS 后检查系统变量中是否添加这个安装路径，方法是：右击“我的电脑”，选择“属性”→“高级”→“环境变量”，查看在弹出的窗口中的“系统变量”的“Path”中是否添加了“G:\Program Files\cvsnt”，如果没有添加或填错，请将该内容正确地添加，它将直接影响 CVSNT 的操作。

7.3　在 Linux 平台上的配置 CVS 服务器

1. 选定 CVS 资源库

CVS 资源库，是 CVS 服务器保存各种软件资源的地方，项目中的所有软件资源和不同版本都可以存放在 CVS 资源库中，关于它的具体概念请参看 7.4 节中 CVS 资源库的概念。下面就选定服务器的一个文件夹作为 CVS 资源库。用户也可以按照自己的习惯和需要来自行选定 CVS 资源库的位置，在这里，选定/cvs/repository 文件夹作为 CVS 资源库。

```
[root@DKD-SV1 cvs]# mkdir /cvs
[root@DKD-SV1 cvs]# mkdir /cvs/repository
```

在 CVS 资源库下，为项目源代码文件创建一个目录。

```
[root@DKD-SV1 cvs]# mkdir /cvs/repository/project/
```

2. 初始化 CVS 服务器

在选定 CVS 资源库后，要进行 CVS 服务器的初始化工作。在初始化过程中，CVS 服务器将创建所需要的系统文件。

```
[root@DKD-SV1 root]# cvs -d /cvs/repository/ init
```

查看一下/cvs/repository/CVSROOT 下的目录和文件：

```
[root@DKD-SV1 root]# ls -a /cvs/repository/CVSROOT/
.                .#taginfo        cvswrappers      notify,v
.#checkoutlist   .#verifymsg      cvswrappers,v    rcsinfo
.#commitinfo     ..               editinfo         rcsinfo,v
.#config         Emptydir         editinfo,v       taginfo
.#cvswrappers    checkoutlist     history          taginfo,v
.#editinfo       checkoutlist,v   loginfo          val-tags
.#loginfo        commitinfo       loginfo,v        verifymsg
.#modules        commitinfo,v     modules          verifymsg,v
.#notify         config           modules,v
.#rcsinfo        config,v         notify
```

3. CVS 服务器权限设定

CVS 资源库建立好了，下一步就是为 CVS 建立用户组和用户，并为他们设定权限。

首先说一下采用的权限机制。在实际工作中通常将 CVS 的使用小组分为两种：CVS 管理小组和 CVS 使用小组。其中，CVS 管理小组的成员负责 CVS 服务器的维护、管理；CVS 使用小组的成员只能使用 CVS 对源代码文件进行版本管理。

在这里，设定两个用户组：cvsmanager 和 cvsuser。其中 cvsmanager 小组是 CVS 管理小组，cvsuser 是使用小组。

（1）创建 CVS 管理小组 cvsmanager：

```
[root@DKD-SV1 root]# groupadd cvsmanager
```

（2）创建 CVS 管理员账号：

```
[root@DKD-SV1 root]# adduser -g cvsmanager cvsadm -p cvsadmabc
[root@DKD-SV1 root]# passwd cvsadm
```

（3）创建使用小组 cvsuser：

```
[root@DKD-SV1 root]# groupadd cvsuser
```

（4）创建用户账号：

```
[root@DKD-SV1 root]# adduser -g cvsuser cvsusera
[root@DKD-SV1 root]# passwd cvsusera
[root@DKD-SV1 root]# adduser -g cvsuser cvsuserb
[root@DKD-SV1 root]# passwd cvsuserb
[root@DKD-SV1 root]# adduser -g cvsuser cvsuserc
[root@DKD-SV1 root]# passwd cvsuserc
```

（5）拥有对配置文件夹/cvs/repository/CVSROOT 设定权限。

CVS 服务器管理小组的成员是 CVS 服务器的管理员，所以拥有对配置文件夹/cvs/repository/CVSROOT 下所有文件和目录的读写权利。同时，使用小组对该文件夹拥有读的权限。

下面，对文件夹/cvs/repository/CVSROOT 的权限进行设定：

```
[root@DKD-SV1 cvs]# chmod 777 /cvs/repository/
[root@DKD-SV1 repository]# chgrp -R cvsmanager /cvs/repository/CVSROOT
[root@DKD-SV1 repository]# chmod -R 075 /cvs/repository/CVSROOT
```

（6）为使用小组追加对/cvs/repository/CVSROOT/history 文件的写权利：

```
[root@DKD-SV1 repository]# chmod 077 /cvs/repository/CVSROOT/history
```

（7）修改/cvs/repository/project 目录的权限：

```
[root@DKD-SV1 CVSROOT]# chgrp -R cvsuser /cvs/repository/project
[root@DKD-SV1 CVSROOT]# chmod 770 /cvs/repository/project
```

4. 设置 CVS 口令服务器

（1）请用超级用户（root）身份，因为本书使用的是 xinetd 的系统，所以需要修改/etc/xinetd.conf 文件，请用以下指令：

```
[root@DKD-SV1 root]# vi /etc/xinetd.conf
```

请将下面内容写到文件中：

```
service cvspserver
{
   port        = 2401
   socket_type = stream
   wait        = no
   user        = root
   server      = CVS可执行路径
   server_args = -f --allow-root=CVS资源库目录 pserver
   log_on_failure +=USERID
   bind - 本机的IP的址
}
```

其中：

① CVS 可执行路径——如果你的机器已经默认安装了 CVS，那么“CVS 可执行路径”就是“/usr/bin/cvs”；如果是自己安装的 CVS 则很有可能是“/usr/local/bin/cvs”。

可以执行下面的语句查找 cvs 可执行路径：

```
[root@DKD-SVI bin]# whereis cvs
cvs: /usr/local/bin/cvs
```

那么，/usr/local/bin/cvs 就是 CVS 可执行路径。

② CVS 资源库目录是建立 CVS 资源库的位置，这里是“/cvs/repository”。

③ 本机的 IP 的址。可以用 ifconfig 查看网络设备，这里是 192.168.0.100。添加的内容如下：

```
service cvspserver
{
   port          = 2401
   socket_type   = stream
   wait          = no
   user          = root
   server        = /usr/local/bin/cvs
   server_args   = -f --allow-root=/cvs/repository pserver
   log_on_failure +=USERID
   bind = 192.168.0.100
}
```

（2）修改/etc/services 文件：

```
vi /etc/services
```

添加如下内容：

```
cvspserver      2401/tcp
```

至此，CVS 口令服务器就配置完成了。

5. 启动 ineted/xinetd 超级服务器，查看 CVS 服务器运行情况

重启 inetd 使修改生效：

```
/etc/rc.d/init.d/xinetd restart
```

至此 CVS 服务器就配置好了，来看一下 CVS 服务器是否已经运行：

```
netstat -lnp|grep 2401
```

若显示类似以下内容，就表明你的 CVS 服务器已经启动成功。

```
tcp       0    0 0.0.0.0:2401          0.0.0.0:*           LISTEN   9830/xinetd
```

这样，CVS 服务器就已经成功地安装到你的机器里了，接下来的工作就是如何运用 CVS 服务器来进行版本的控制了，这部分内容将在下面的章节中做详细介绍。

7.4 在 Windows NT 平台上的配置 CVS 服务器

本节介绍 CVS 服务器在 Window NT 上的配置。

1. 查看 CVS 服务器状态

选择“开始”→“程序”→“CVSNT”→“Service control panel”。如图 7-1 所示，CVS 服务器和 CVS Lock 服务器都已经启动起来了。

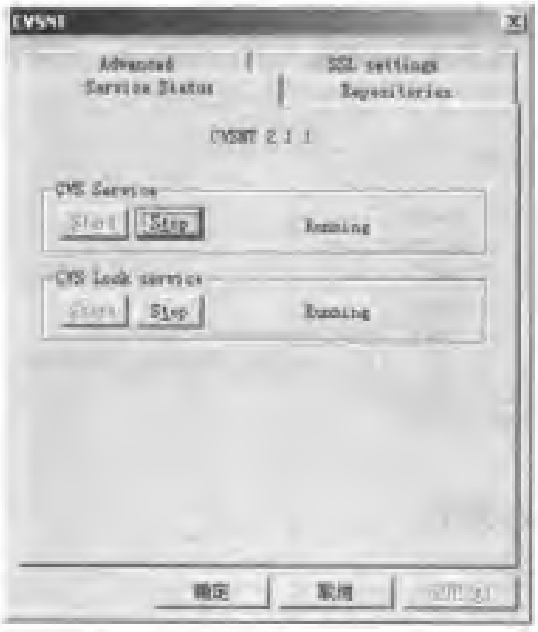


图 7-1 CVS 服务器和 CVS Lock 服务器启动

2. 建立 CVS 资源库

选择“Repositories”选项，在该窗口添加后编辑 CVS 资源库，单击“Add”，在弹出的窗口中添加 CVS 资源库的地址。这里，选择 D:\project。

3. 选择连接方式

选择“Advanced”选项，选择“Use local user for pserver authentication instead of domain users”，在“Temporary”选项中选择错误等信息的输出路径。

这样，就配置好了 CVS 服务器及其资源库。在下面的章节中将介绍如何通过 Eclipse 连接 CVS 服务器进行版本控制。

7.5 应用 CVS 系统协助开发的一个简单的流程

在 7.4 节里，配置了 CVS 服务器。在图 7-2 中，列出了一个利用 CVS 系统进行版本管理的简单的流程。

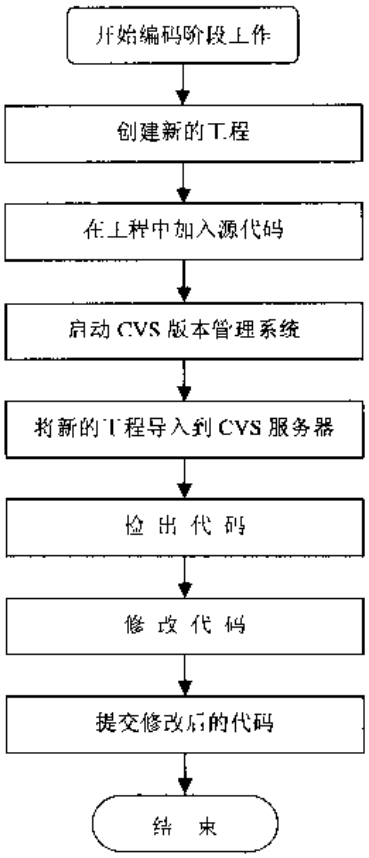


表 7-2　利用 CVS 系统进行版本管理的简单的流程图

下面对该流程进行简单的解说。在多人组成的项目中，通常的情况是开发小组的成员都共用一个 CVS 服务器，有共同的版本库，每个开发人员的计算机相对于 CVS 服务器都是一个客户端。当项目进入到编码阶段时，首先，由项目组的一个成员在自己的计算器上建立一个项目，然后将它提交到 CVS 服务器上，这时 CVS 系统就会在相应的 CVS 仓库中建立一个模块，该模块用来存放该项目。从这时起，CVS 版本管理系统启动，其他的开发人员与 CVS 资源库进行连接后，向 CVS 服务器提出取得工程文件的要求，CVS 服务器在验明该用户的身份后，向用户发送一个工程文件的备份，客户端接到这个备份后，将备份保存到相应的目录下。从资源库中取出工程的文件（这个过程称为检出代码或建立工作备份）。

当开发人员将修改后的源代码提交到 CVS 资源库时，CVS 服务器会检验该源代码是否是用最新版本修改的，如果不是用最新版本修改，将拒绝将该源代码保存为最新的版本；反之，则将该源代码文件作为最新版本的源文件。值得注意的是 CVS 系统并不是将每次提交的源文件都保存下来，而是将版本间的不同保存起来，这样大大节省了物理空间。

7.6 CVS 的几个基本概念

接下来了解一下 CVS 版本控制系统的一些基本的概念，只有真正了解了 CVS 的基本结构和基本的工作原理，才能有效地利用 CVS 版本控制系统，从而高效地完成软件项目的开发。

1. CVS 版本控制系统的系统结构

CVS 版本控制系统的系统结构采用的是 Client/Server（客户端/服务器）结构。

CVS 版本控制系统结构工作的过程是这样实现的：CVS 用户在客户端上工作，要进行 CVS 的操作时，首先，连接 CVS 服务器，向 CVS 服务器提出 CVS 命令，CVS 服务器得到请求后，按照 CVS 客户端提出的命令执行相应的操作，并向 CVS 客户端发出操作结果。

2. CVS 资源库（CVS repository）

CVS 资源库，也称 CVS 资源库或主备份，是 CVS 服务器保存各种软件资源的地方，项目中的所有软件资源和不同版本都可以存放到 CVS 资源库中。

在多人组成的项目中，通常的情况是开发小组的开发人员都共用一个 CVS 服务器，有共同的版本库。每个开发人员自己的机器相对于 CVS 服务器都是一个客户端，在项目开始时，由项目组的一个成员在自己的计算器上建立一个项目，然后将它提交到 CVS 服务器上，这时 CVS 系统就会在相应的 CVS 仓库中建立一个模块。其他的开发人员与 CVS 资源库进行连接，之后从资源库中取出项目的文件（这个过程称为检出代码或建立工作备份）。

3. 工作备份（working copy）

工作备份是在项目开发小组中每一个开发人员在进行项目开发时都拥有一个与其他小组成员相互独立的全部源代码文件的备份。在 CVS 服务器中项目的所有资源都保存在 CVS 资源库中，当一个开发人员要对源代码进行修改时，就需要从 CVS 服务器上下载所有项目的所有资源，之后，在自己的机器上修改源文件。在这个过程中，开发人员从 CVS 服务器上下载的文件就是 CVS 服务器给该客户端提供的一个工作备份。开发人员在客户端修改工作备份的内容时，不会修改 CVS 服务器上的文件，更不会影响到其他小组成员的工作备份。

4. 检出代码（创建工作备份）

检出代码的过程就是创建工作备份的过程。当从 CVS 服务器中取出源文件，CVS 服务器会检查每一个源文件，并把它们交给客户端，在客户端就生成了一份工作备份。

5. 冲突（conflict）

在项目组协同开发的情况下，当两个程序员对同一个文件的同段代码进行修改后，依次向 CVS 服务器提交文件，这时就会发生冲突。这种情况下，虽然系统会给出冲突标记，但是需要开发人员手工将冲突部分修改好，修改后将文件向 CVS 服务器提交。

6. 提交（commit）

将写好的代码送到服务器上保存的过程就是提交。开发人员在写好的代码后将代码文件向 CVS 服务器传送，CVS 服务器会检测提交的文件是否与 CVS 资源库中的文件发生冲突，如果没有发生冲突，就按照该代码文件生成最新版本的代码文件。

7. 更新（update）

在多人协同开发的情况下，有的代码文件可能被两个人或多个人修改。这种情况下，修改该代码文件时就需要先从 CVS 服务器上更新最新版本的代码文件。

8. 模块

将建立好的一个工程提交到 CVS 服务器，该工程的所有文件组成一个模块。

7.7　Eclipse 和 CVS 的连接

在 7.6 节中，读者了解了几个基本的 CVS 概念，这是应用 CVS 版本管理系统进行版本管理的基础，本节将介绍 Eclipse 和 CVS 的连接，开始 Eclipse 和 CVS 协同开发项目的完美结合。

1. “CVS 资源库”透视图（CVS Repository Exploring）

Eclipse 平台拥有内置 CVS 客户机，它是作为一个单独透视图而实现的，用于与 CVS 服务器的交互。图 7-3 即 CVS 资源库透视图的全貌。

打开“CVS 资源库”透视图的方法：依次选择“窗口”→“打开透视图”→“CVS 资源库浏览”。

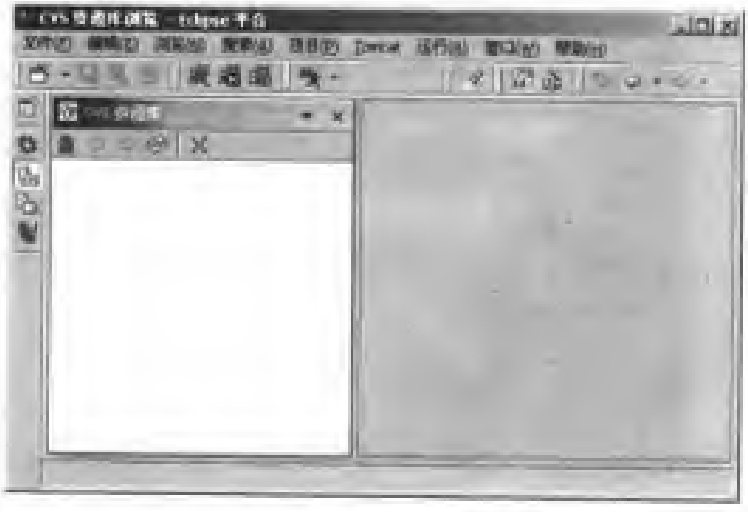


图 7-3　“CVS 资源库”透视图

2. 连接 CVS 服务器资源库的位置

打开“CVS 资源库”透视图，其中显示的是已有的连接，由于现在还没有建立与 CVS 服务器资源库的连接，所以，在“CVS 资源库”透视图中没有任何项目可以选择。

接下来利用配置好的CVS服务器来建立新的CVS服务器资源库连接。

“CVS资源库”透视图中右击鼠标（见图7-4），单击“新建”→“资源库位置”。

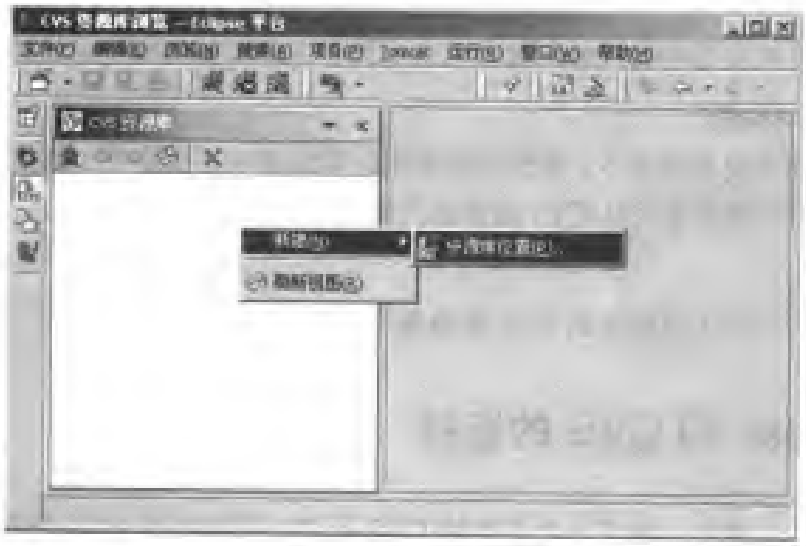


图7-4 建立新的CVS服务器资源库连接

在弹出的“添加CVS资源库”窗口（见图7-5），添加资源库的位置以及登录信息。

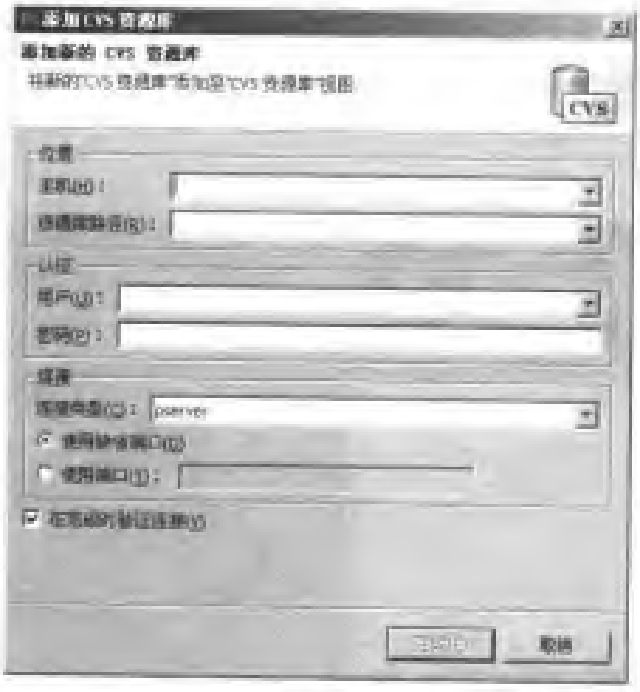


图7-5 “添加CVS资源库”窗口

下面对窗口的各个项目进行解释。

（1）主机：CVS服务器的IP地址。

（2）资源库路径：CVS资源库的所在位置。

（3）用户：进行连接的用户名。

（4）密码：连接用户的密码。

（5）连接类型：选择资源库的 CVS 连接类型，默认为 pserver 连接。

（6）使用默认端口或使用端口。

在 Linux 环境下配置的 CVS 服务器的资源库的连接情况如图 7-6 所示，在 Window NT 环境下配置的 CVS 服务器的资源库的连接情况如图 7-7 所示。

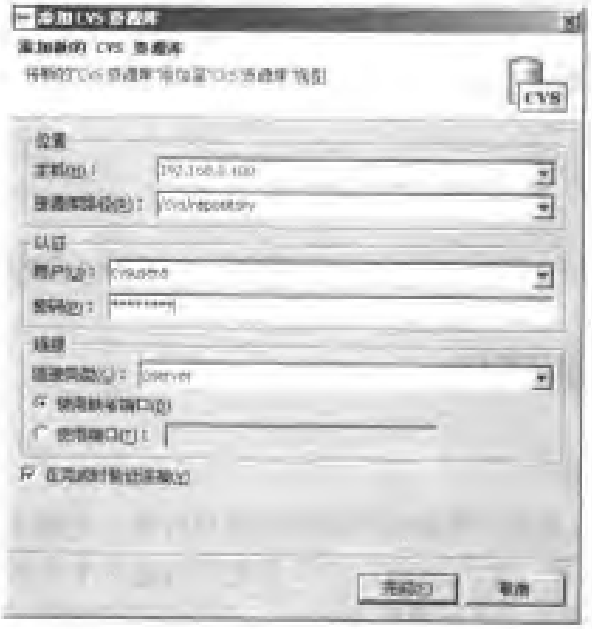


图 7-6　Linux 环境下 CVS 服务器的资源库的连接情况



图 7-7　Window NT 环境下配置的 CVS 服务器的资源库的连接情况

单击“完成”，与 CVS 资源库的连接就建立好了。此时，在“CVS 资源库”透视图中就会出现配置好的 CVS 服务器的资源库了，如图 7-8 所示。



图 7-8 透视图中就会出现配置好的 CVS 服务器的资源库

7.8 启动一个新项目

1. 启动一个新项目

CVS 版本管理系统是用来帮助实现软件的项目管理。使用 CVS 的目的是如何有效地进行源代码文件的管理。所以如何在 CVS 系统中创建一个全新的项目，以及如何将已有的项目导入 CVS 系统是一个很重要的部分。它是有效地利用 CVS 版本管理系统进行项目管理的开始。

2. 创建一个新项目，并将该项目导入到 CVS 服务器

利用 Eclipse 创建一个新的 Java 项目，名字为 helloCVS。创建项目后，就要将这个崭新的项目添加到 CVS 服务器中，方法是（见图 7-9）：在“包资源管理器”中 helloCVS 上右击鼠标，单击“小组”→“共享项目”。

在弹出“与 CVS 资源库共享项目”窗口中选择项目要导入的 CVS 资源库，如图 7-10 所示。

（1）如果没有建立 CVS 资源库，选择“创建新的资源库位置”，并请参考上一节的内容建立新的 CVS 资源库。

（2）如果已经建立了 CVS 资源库，则选择“使用现有的资源库位置”，并选择一个 CVS 资源库作为项目要导入的 CVS 资源库。

单击“下一步”，出现“输入模块名”窗口（见图 7-11），在这个窗口中可以创建 CVS 资源库的模块名，其实，CVS 系统中的模块就是指项目。

（1）如果想用项目名作为模块名，就选择“使用项目名作为模块名”。

（2）如果想自己给模块起个名字，就选择“使用指定的模块名”，在文本框中填写模块保存的路径名（默认为 CVS 资源库所在的目录）和模块名。



图 7-9　将新建项目导入到 CVS 服务器

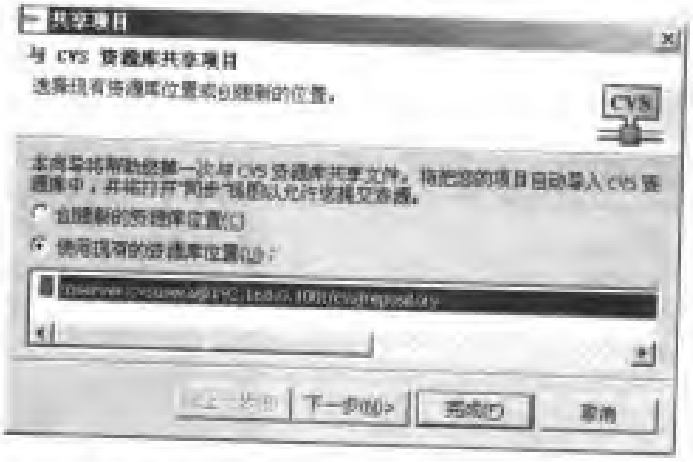


图 7-10　选择项目要导入的 CVS 资源库

选择“使用指定的模块名”，并在文本框中写入：project/helloCvs_01。其中：project 是模块保存的路径名；helloCvs_01 是模块名。之后，单击“下一步”。



图 7-11 “输入模块名”窗口

在“准备好共享项目”窗口中（见图 7-12），单击“完成”按钮。

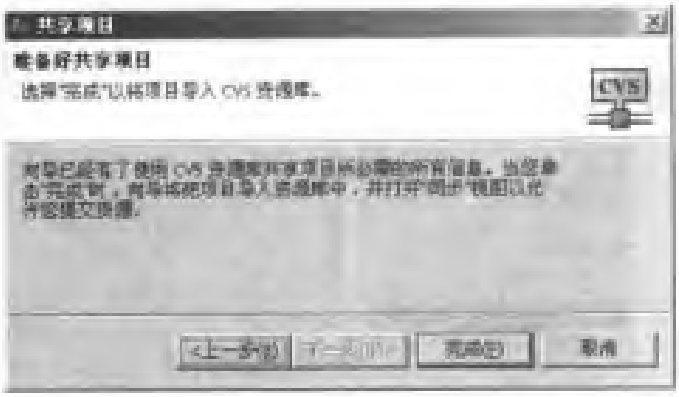


图 7-12 “准备好共享项目”窗口

这样，新的项目就添加到 CVS 服务器中了，也就意味着该项目已经启动 CVS 版本管理系统了。可以切换到“CVS 资源库”透视图中，看一下共享的项目是否导入到 CVS 资源库中了。如图 7-13，项目已经导入 CVS 资源库了。

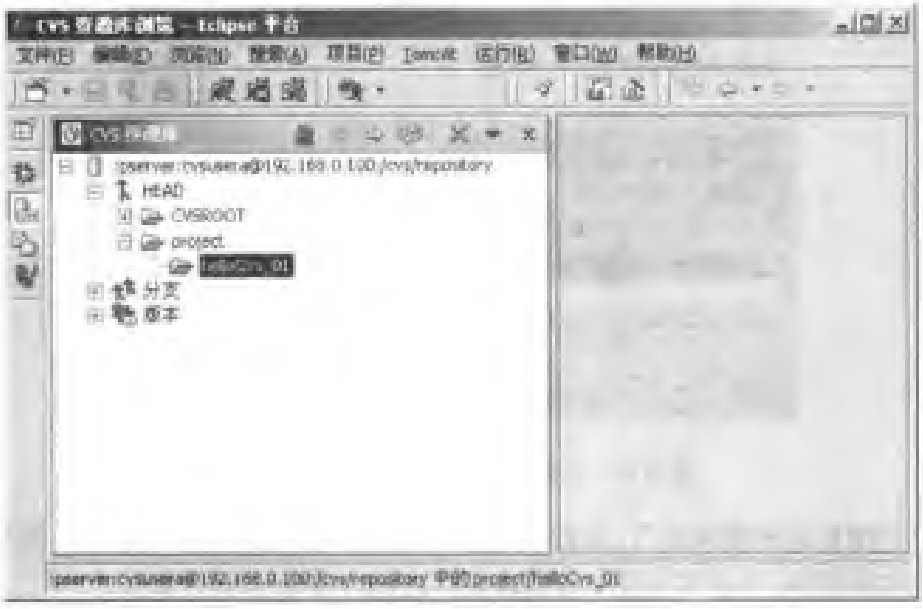


图 7-13 新建项目出现在 CVS 资源库

7.9　应用 Eclipse 从 CVS 资源库检出代码

在上一节中，用 Eclipse 新建了一个项目并将该项目导入到 CVS 资源库中。在一个项目组中，一个成员建立好项目，这时，在 CVS 资源库中就有了该项目的内容，项目组的其他成员要开发该项目，就需要从 CVS 服务器上将项目导入到自己的计算机上，这个过程就是检出代码。

7.8 节中的项目 helloCvs 是由开发人员 cvsusera 建立的，那么与 usera 同组的开发人员 userb 要想与 usera 一同开发 helloCvs 项目，就必须将项目 helloCvs 导入到自己的计算机中，也就是说 userb 必须要检出代码。

将一个项目从 CVS 资源库导入客户端的方法是如下。

（1）与 CVS 资源库建立连接（该步骤请参照 7.7 节“Eclipse 和 CVS 的连接”），如果已建立与 CVS 资源库的连接则可以跳过这一步，直接进入步骤（2）。

（2）方法①：切换到“CVS 资源库”透视图，在“CVS 资源库”透视图中，找到想要导入的模块，用鼠标右击该模块，在下拉菜单中选择“作为项目检出”，如图 7-14 所示。



图 7-14　选择“作为项目检出”

执行上面的步骤后，切换到“包资源管理器”视图，就会发现项目 helloCvs 已经出现，如图 7-15 所示。此时的项目名变成了 helloCvs_01，这是因为 CVS 服务器给客户端发送工作备份时，给出的是 CVS 资源库的模块名。



图 7-15　项目 helloCvs 出现在“包资源管理器”视图中

方法②：切换到“CVS 资源库”透视图，在“CVS 资源库”透视图中，找到要导入的模块，用鼠标右击该模块，在下拉菜单中单击“检出为”，如图 7-14 所示。选择后，弹出选择项目种类的窗口（见图 7-16），在右侧的窗口中选择要生成 Java 项目的种类。



图 7-16　选择要生成 Java 项目的种类

在这里，选择“Java 项目”，单击“下一步”，见图 7-17，在“项目名”文本框中，写上项目的名称。在这里添入“helloCvs”，单击“下一步”。

在弹出的窗口中，见图 7-18，设置 Java 项目的各个选项，设定好后单击“完成”按钮。

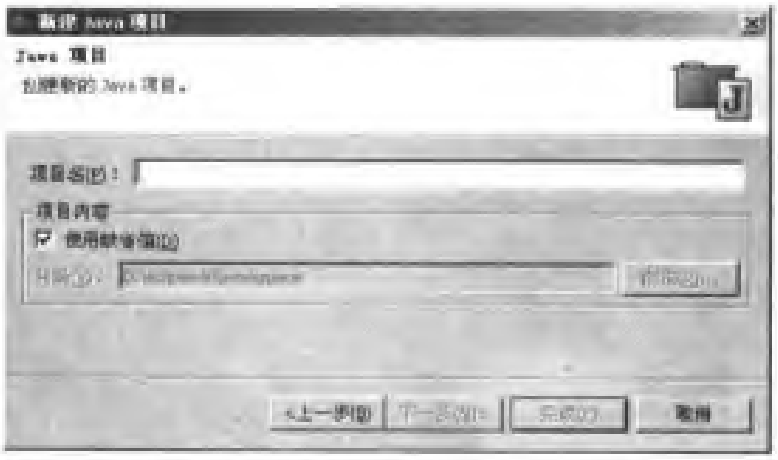


图 7-17　“项目名”文本框中，写上项目的名称



图 7-18　设置 Java 项目的各个选项

弹出“确认透视图切换”的对话框，见图 7-19，询问是否要切换到 Java 透视图。



图 7-19　“确认透视图切换”的对话框

单击“是”后，在“包资源管理器”中就出现刚才导入的 Java 项目 helloCvs 了，如图 7-20 所示。

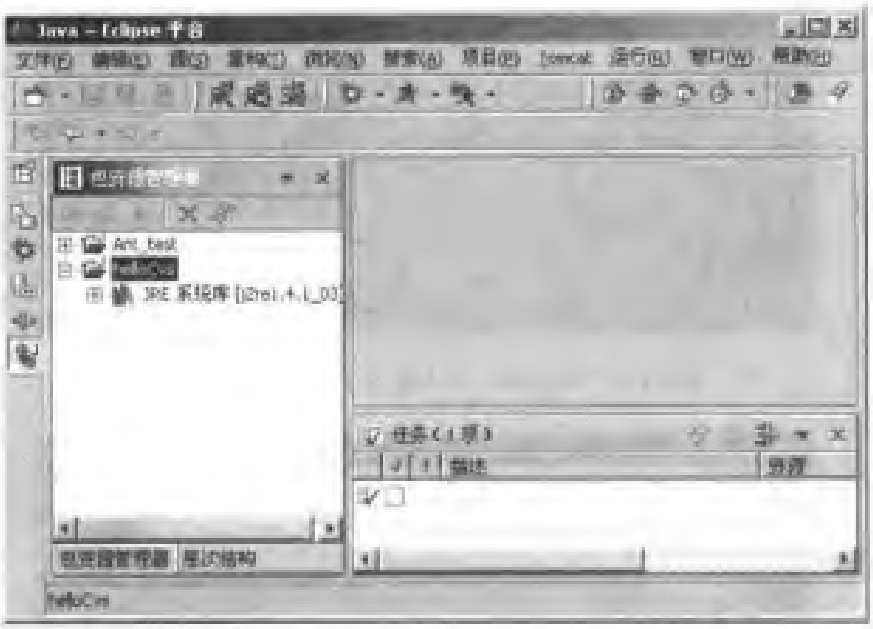


图 7-20 “包资源管理器”中出现导入的 Java 项目 helloCvs

方法③：见图 7-14，在下拉菜单中单击“检出到”，选择该选项可以将所选项目导入已建好的项目下。

7.10 在开发过程中 Eclipse 协同 CVS 开发的基本操作

1. 提交

将写好的源代码送到 CVS 服务器上保存的过程就是提交。

现在，由开发人员 usera 在 Eclipse 中新建一个类，类名为 CvsTestSample，源代码如下所示。

```
package com.dkd.cvstest;

public class CvsTestSample {

        private String s_address = "";

        /* Getting address
        */
        public String getAddress() {
                return s_address;
        }

        /*Setting address
        */
        public void setAddress(String s_address) {
```

```
            this.s_address = s_address;
        }

    } // the end -1*
```

下面把这个代码文件提交到 CVS 资源库中，方法是：右击类 CvsTestSample→“小组”→“提交”，如图 7-21 所示。



图 7-21　提交

然后会出现“添加至 CVS 版本控制”窗口，如图 7-22 所示。在窗口中会出现一句话“存在不受 CVS 版本控制的 1 资源。想要添加它吗？”，因为在 CVS 资源库中现在并没有 CvsTestSample 类及其所在包的文件夹，所以，CVS 系统提示是否要添加该源代码文件及其所在目录。

在“添加至 CVS 版本控制”窗口选择“是”后，会出现“提交”窗口，如图 7-23 所示。



图 7-22 “添加至 CVS 版本控制”窗口

（1）在“编辑提交注释”的文本框中，可以编写你的提交注释。

（2）在“选择先前输入的注释”中，可以将你写过的提交注释添加到“编辑提交注释”的文本框。



图 7-23 “编辑提交注释”文本框

在“添加至 CVS 版本控制”窗口选择“确定”后，新编写的类 CvsTestSample 就会添加到 CVS 资源库了。请切换到“CVS 资源库”透视图。在“CVS 资源库”透视图中，用鼠标右击刚才建好的 CVS 资源库连接，单击“刷新视图”，然后展开 HEAD → project → CvsTest_01 → com → dkd → cvstest，新编写的类 CvsTestSample 就会出现，如图 7-24 所示，CvsTestSample.java 1.1 表示源文件 CvsTestSample.java 目前在 CVS 资源库的最新版本是 1.1 版，在后续的介绍中会出现该版本的变更。

2. 更新

从 CVS 服务器上获取最新版本的源代码文件并用该文件覆盖客户端的源代码文件，此过程称为更新。

在 7.9 节中，开发人员 userb 已经将项目 HelloCVS 导入到自己的计算机里了。如果他要对 usera 建立的类 CvsTestSample 进行修改，首先，需要将类 CvsTestSample 更新到自己计算机里。方法是：鼠标右击项目名→“小组”→“更新”，如图 7-25 所示。



图 7-24　CvsTestSample.java 1.1



图 7-25　选择“更新”选项

3. CVS 资源库的内容进行比较

开发人员 cvsusera 在类 CvsTestSample 中，添加了一个方法 cntPersonNum，语句如下。

```
Package com.dkd.cvstest;
public class CvsTestSample {
        private String s_address = "";
        /* Getting address
        */
        public String getAddress() {
                return s_address;
        }
        /*Setting address
        */
        public void setAddress(String s_address) {
                this.s_address = s_address;
        }
        /* Count personNum
        */
        public void cntPersonNum() {
    }
} // the end -1^
```

用鼠标右击类 CvsTestSample →“比较”→“HEAD 的最新内容”。至此 CVS 比较窗口的内容如图 7-26 所示。



图 7-26 “CVS 比较”窗口

在“结构比较”窗口，双击“CvsTestSample.java（ASCII-kkv）”，会出现图 7-27 的情况。其中，“Java 源代码比较”是 cvsusera 本机上修改后的代码，而“资源库文件”给出的是 CVS 资源库的代码。两者不同之处只是 cvsusera 在类 CvsTestSample 中，添加了一个方法 cntPersonNum，Eclipse 将不同之处用蓝色涂满，并用线连接了起来。

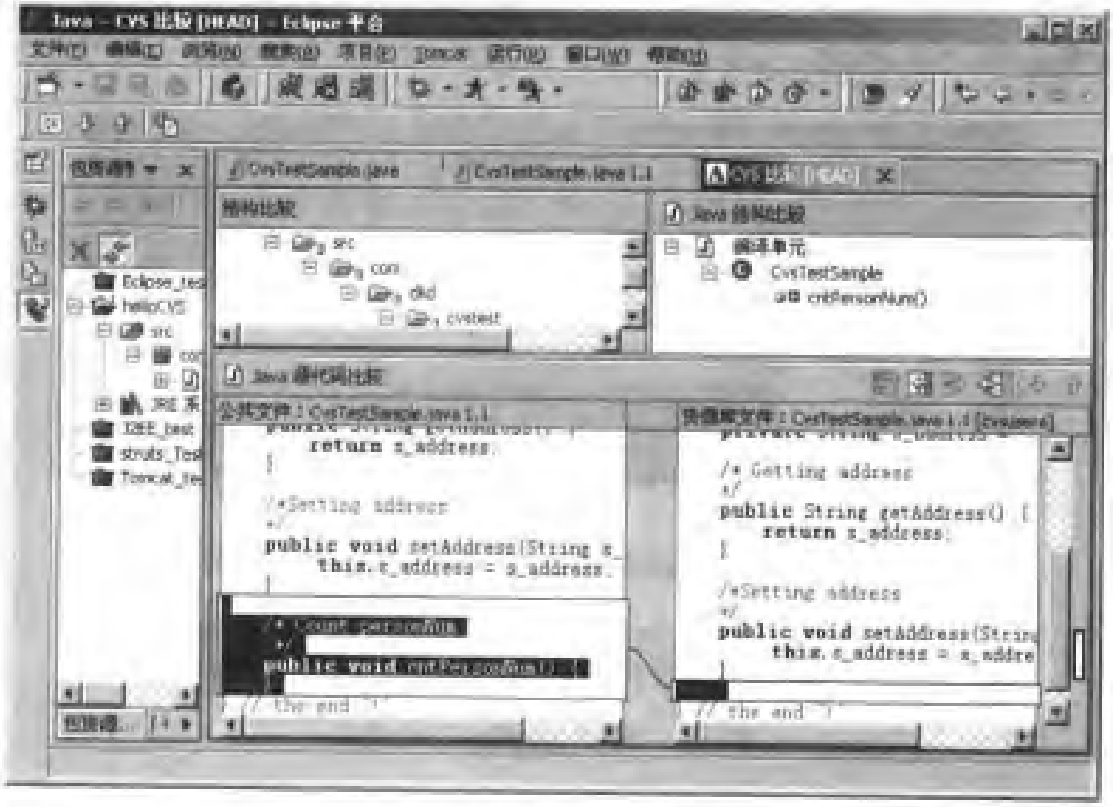


图 7-27　“结构比较”窗口

用这种比较的方法使开发人员很清楚的看出自己的源代码与 CVS 资源库中当前最新版本的不同之处。

4. 在资源历史记录中显示

大多数开发人员都遇到过类似这样的问题：一个程序已进行了一段时间的开发，突然，用户决定取消一部分功能。而这部分功能是这几天才做好了的，项目经理希望你能用几天前的版本解决这个问题，可是不凑巧，你没有作备份。但是，要知道用户的需求是常有的，而需求变更是很正常的。如果使用了 CVS 版本管理系统，这个问题就迎刃而解了。

下面举例说明，将刚才 cvsusera 修改后的类提交后，CVS 资源库中 CvsTestSample 类的版本就是 CvsTestSample 1.2 了。

切换到“包资源管理器”，用鼠标右击 CvsTestSample 类→“小组”→“在资源历史记录中显示”，如图 7-28 所示。

窗口“CVS 资源历史记录”中显示的就是软件开发过程中各个不同版本。其中：

- 修订——版本号，含有“*”字的是开发人员当前开发的版本；
- 标记——被符号化了的版本；
- 日期——版本被创建的日期；
- 作者——登录 CVS 服务器的用户名；
- 注释——提交版本前的注释。

只需要选择一个版本，用右键单击它。在下拉框中，如果选择“获取内容”则该版本将覆盖本机当前的内容；如果选择“打开”，则将会把这个版本打开。



图 7-28 “CVS 资源历史记录”窗口

本章小结

在本章中，读者了解了 CVS 系统、CVS 服务器的安装与配置，并介绍了 Eclipse 和 CVS 协同开发程序的简单操作。读者只要掌握了这些技巧，就可以运用 Eclipse 进行常用的 CVS 操作。

CVS 版本管理系统已经成为了一个主流的版本管理系统，要想真正掌握它，需要仔细体会它的设计理念。编者相信，通过努力地探索你会更加感受到它的庞大和精妙，只有这样才会灵活和有效的运用它，使你的项目能按照既定计划稳步地开发。

第五篇

Eclipse 开发工具

PART 5

第 8 章　Web 应用程序开发

第 9 章　使用 Eclipse 进行 J2EE 项目开发

第 8 章　Web 应用程序开发

随着网络技术的飞速发展，Web 应用程序开发已经在应用程序开发中占据了主导地位，本章介绍通过在 Eclipse 上安装 Tomcat 服务器插件的方法，进行 Web 应用程序的开发，将讲解以下内容：

- Web 服务器的介绍
- Tomcat 服务器的介绍、下载和安装
- 使用 Tomcat 服务器插件与 Eclipse 平台的连接
- 如何创建 Tomcat 项目
- Tomcat 项目中的 Servlet 开发和部署
- Tomcat 项目中的 JSP 的开发

8.1　Web 服务器

Web 服务是 Internet 服务器最基本的服务，编写一个基本的 Web 服务器是非常简单的，仅仅需要监听合适的端口、建立连接、发送数据。编写这样的服务器一般和客户端的开发总是相辅相成地发展的。

当 Web 服务成为很多商务应用都必须面对和采用的技术时，就出现了很多不同 Web 服务器以满足这些不同的需求。本节介绍几种运行 Servlet 时常用的 Web 服务器，及 Web 服务器的安装配置。

8.1.1　Web 服务器简介

1．Apache 服务器

全球应用最广泛的 Web 服务器软件就是 Apache，它是世界排名第一的 Web 服务器，根据著名的 Web 服务器调查公司 Netcraft 的调查，世界上 50%以上的 Web 服务器都在使用 Apache。Apache 不但具有庞大的用户群基础，并且有丰富的配置使用文档。

Apache 提供的功能包括：目录索引、目录别名、内容协商、可配置的 HTTP 错误报告、CGI 程序的 SetUID 执行、子进程资源管理、服务器端图像映射、URL 重写、URL 拼写检查以及联机手册等。

Apache 对于 Linux 用户来说，也是最容易使用的 Web 服务器，而且它还是免费的。如果读者仅仅是想要用 Apache 提供基本的 Web 页面服务，那么你可能根本不需要调整任何配置。想了解更多信息可以访问 Apache 的官方网站：http://jakarta.apache.org/。

2. Tomcat 服务器

众所周知，Tomcat 是一个免费的开放源码的 Serlvet 容器，它是 Apache 基金会的 Jakarta 项目中的一个核心项目，也是 Sun 公司官方推荐的 Servlet 和 JSP 容器。Servlet 和 JSP 的最新规范都可以在 Tomcat 的新版本中得到实现。Tomcat 具有轻量级和灵活嵌入到应用系统中的优点，所以得到了广泛的应用。在 Tomcat 的发展中，Sun 公司在 1999 年 6 月宣布参与 Jakarta 项目的 Tomcat Servlet 容器和 JSP 引擎的开发，使得 Tomcat 在 3.x 和 4.x 版之间系统设计上发生了比较大的变化。Tomcat 现在的最新版本是 5.x，不过最常用的版本仍然是 4.x，读者可以访问 http://jakarta.apache.org/网站，以得到更多的信息。

3. Tomcat 与 Apache 的整合

一般情况下，在 Linux 服务器端都会配置与 Tomcat Servlet 容器集成在一起 Apache Web 服务器，这种配置的优势在于把 Tomcat 在管理 Servlet 方面的优势和 Apache 在提供静态内容方面的优势有机地结合起来。Apache 被配置为监听默认端口 80 的 Web 请求，Tomcat 用于监听定制端口（默认端口号为 8080）的 Servlet 请求。

8.1.2 在 Linux 下安装 Apache 和 Tomcat

1. 所需的软件包

- j2sdk-1_4_2_01-linux-i586-rpm.bin
- httpd-2.0.40-i686-pc-linux-gnu-rh73.tar.gz
- jakarta-tomcat-4.1.30.tar.gz
- jakarta-tomcat-connectors-4.1.30

2. 创建保存安装文件的文件夹

```
mkdir -p /usr/local/src/archive/apache
mkdir -p /usr/local/src/archive/tomcat
mkdir -p /usr/local/src/archive/j2sdk
mkdir -p /usr/local/src/archive/mod
```

3. 下载安装所需要的文件

（1）Apache2.0.40 的下载地址是 http://httpd.apache.org，将 httpd-2.0.40-i686- pc-linux-gnu-rh73.tar.gz 文件保存在/usr/local/src/archive/apache 目录下面。

（2）j2sdk-1_4_1_03 的下载地址是 http://java.sun.com/j2se/1.4.1/download.html，将 j2sdk-1_4_1_03-linux-i586-rpm.bin 文件保存在目录/usr/local/src/archive/j2sdk 下。

（3）Tomcat 4.1.30 的下载地址是 http://jakarta.apache.org/builds/，将 tomcat- 4.1.30. tar.gz 文件保存在目录 /usr/local/src/archive/tomcat 下面。

（4）jakarta-tomcat-connectors-4.1.30 的下载地址是 http://mirrors.isc.org/pub/apache/jakarta/tomcat- 4/v4.1.30/src/，文件保存在目录 /usr/local/src/archive/mod 下面。

4. 解压下载的软件包

```
cd /usr/local/src/archive/apache
```

```
tar zxvf httpd-2.0.40-i686-pc-linux-gnu-rh73.tar.gz
cd /usr/local/src/archive/tomcat
tar zxvf tomcat4.1.30.tar.gz
cd /usr/local/src/archive/mod
tar zxvf jakarta-tomcat-connectors-4.1.30-src.tar.gz
```

5. 安装 j2sdk

```
cd /usr/local/src/archive/j2sdk
chmod 755 j2sdk-1_4_2_01-linux-i586-rpm.bin
./j2sdk-1_4_2_01-linux-i586-rpm.bin
```

当系统提问 yes or no 的时候，键入 yes。

```
rpm -ih j2sdk-1_4_2_01-linux-i586.rpm
```

6. 安装 apache_2.0.40

（1）安装 apache_2.0.40

```
mkdir -p /usr/local/apache
cd /usr/local/src/archive/apache/httpd-2.0.40
./configure --prefix=/usr/local/apache --enable-shared=all
make
make install
```

（2）编译 Jakarta-Tomcat-connectors

```
cd /usr/local/src/archive/mod/jakarta-tomcat-connectors-4.1.30-src/webapp
./support/buildconf.sh
./configure --with-apxs
make
```

注释：生成的 mod_webapp.so 文件在 webapp/apache-2.0 目录中。

注意，如果生成 mod_webapp.so 时失败，并提示找不到 apxs，则证明在安装 Apache 的时候，未安装 Apache 附加的开发包，请下载安装该开发包，则会解决该问题。

（3）把 apache-2.0 目录中的 mod_webapp.so 文件复制到目录/usr/local/apache/modules/下：

cp/usr/local/src/archice/mod/jakarta-tomcat-connectors-4.1.30-src/webapp/apache-2.0/mod_webapp.so /usr/local/apache/modules/

并检查文件/usr/local/apache/modules/mod_ webapp.so 的属性，应与其他 Modules 文件属性一致，如果不相同，则使用命令 chmod，chgrp 或 chown 来修改。

7. 安装 Tomcat 4.1.30

```
mkdir -p /usr/local/tomcat
mv /usr/local/src/jakarta-tomcat4.1.30/* /usr/local/tomcat
```

8. 设置环境变量

```
cd /root
vi ~/.bash_profile
```

在文件中追加如下内容：

```
export JAVA_HOME=/usr/java/j2sdk1.4.1_03
export CLASSPATH=$JAVA_HOME/lib:$JAVA_HOME/jre/lib
export PATH=$JAVA_HOME/bin:$PATH
export CATALINA_HOME=/usr/local/tomcat/
```

保存文件后运行：

```
source ~/.bash_profile
```

9. 配置 Tomcat

修改 server.xml 文件：

```
vi /usr/local/tomcat/conf/server.xml
```

将原文件内容作如下修改：

（1）在文件中搜索“Tomcat Examples Context”并在搜索到的位置添加如下内容：

```
<Context path="/Tomcat_test" docBase="/usr/users/Tomcat_test " debug="0"
     reloadable="true" crossContext="true">
     <Logger className="org.apache.catalina.logger.FileLogger"
     prefix="localhost_tomcattest_log." suffix=".txt" timestamp="true"/>
</Context>
```

（2）在文件的最后，结束标签</Server>的上边添加如下内容：

```
<!-- Define an Apache-Connector Service -->
<Service name="Tomcat-Apache">
   <Connector className="org.apache.catalina.connector.warp.WarpConnector"
      port="8008" minProcessors="5" maxProcessors="75" enableLookups="true"
      appBase="webapps" acceptCount="10" debug="0" />
   <Engine className="org.apache.catalina.connector.warp.WarpEngine"
      name="Apache" debug="0">
      <Logger className="org.apache.catalina.logger.FileLogger"
         prefix="apache_log." suffix=".txt"    timestamp="true" />
      <Realm className="org.apache.catalina.realm.MemoryRealm" />
   </Engine>
</Service>
```

10. 配置 Apache

修改 httpd.conf 文件：

```
vi /usr/local/apache/conf/httpd.conf
```

将原文件内容作如下修改：

（1）在文件中找到 ServerName，去掉前面的注释符号（#）：

```
#ServerName new.host.name:80 → ServerName www.TomcatServer.com:80
```

其中，www.TomcatServer.com:80 为 Web 服务器的主机名。

（2）在文件中找到 NameVirtualHost，去掉前面的注释符号（#）：

```
#NameVirtualHost * → NameVirtualHost *
```

（3）在 Dynamic Shared Object (DSO) Support 这一项中追加如下内容：（加载 mod_

webapp.so 模块）

```
LoadModule webapp_module modules/mod_webapp.so
```

（4）在 Bring in additional module-specific configurations 这一项中追加如下内容：（用来与 Tomcat 联动）

```
<IfModule mod_webapp.c>
    WebAppConnection warpConnection warp localhost:8008
    WebAppDeploy /usr/users/Tomcat_test warpConnection /Tomcat_test/
</IfModule>
```

（5）在 VirtualHost 这一项中追加如下内容：

```
<VirtualHost *>
    DocumentRoot /usr/users/Tomcat_test
    ServerName www.Tomcat_test.com
    ErrorLog  /usr/users/Tomcat_test/logs/error_log
    CustomLog  /usr/users/Tomcat_test/logs/access_log common
</VirtualHost>
```

至此 Apache 的安装和配置完成。

11. 启动 Tomcat

```
/usr/local/tomcat/bin/startup.sh
```

12. 启动 Apache

```
/usr/local/apache/bin/apachectl start
```

13. 在客户端设置 Web 服务器的 DNS

打开客户端 C:\WINDOWS\system32\drivers\etc 目录下 hosts 文件，在 hosts 文件将 Web 服务器的 IP 地址与主机名匹配起来。例如，Web 服务器的 IP 地址是 192.168.0.100，在 bosts 文件的后面加上如下内容：

```
192.168.0.100 www.TomcatServer.com
```

14. 启动 IE

键入 URL：http://TomcatServer.com/Tomcat_test/index。
至此所有的安装配置全部完成。

15. 调试

如果启动时出现故障，可以尝试修改 Red Hat 的 iptables 服务：

```
vi/etc/sysconfig/iptables
```

在文件中搜索“COMMIT”，在它的上面添加：

```
##HTTP
-A RH-Lokkit-0-50-INPUT -p tcp -m tcp --dport 80 --syn -j ACCEPT
```

保存退出后，重起 iptables 的服务：

```
service iptables restart
```

16. 关闭和重启命令

关闭 Tomcat:

```
/usr/local/tomcat/bin/shutdown.sh
```

关闭 Apache:

```
/usr/local/apache/bin/apachectl stop
```

重启动 Tomcat:

```
/usr/local/tomcat/bin/startup.sh
```

重启动 Apache:

```
/usr/local/apache/bin/apachectl start
```

8.1.3 在 Windows 下安装 Tomcat

在 Windows 下进行 Tomcat 的安装要比在 Linux 环境下安装简单得多。本机的操作系统是 Windows 2000。在 Tomcat 安装前，要确保你的机器上已经安装上了 Java 开发包（JDK），请参考本书第 1 章的 j2sdk1.4.1_03 下载地址，本机的安装路径为 C:\j2sdk1.4.1_03。

Tomcat 4.1.30 的安装文件下载网址：http://jakarta.apache.org/site/binindex.cgi，执行 jakarta-tomcat-4.1.30.exe 安装文件，选择指定的目录，本机安装 Tomcat 4.1.30 的指定目录为 C:\tomcat4.1，如图 8-1 所示。安装时会自动寻找你的 jdk 和 jre 的位置。

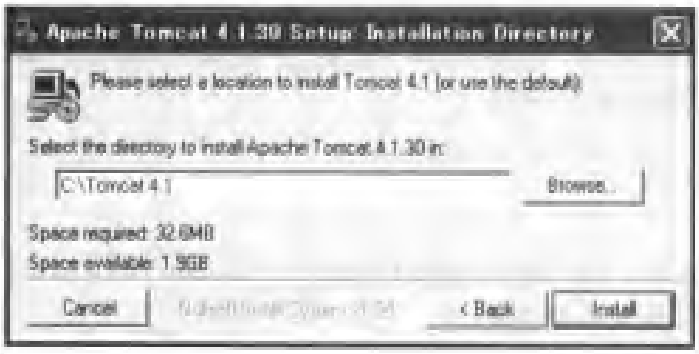


图 8-1 安装 Tomcat 指定路径

安装好 Tomcat 4.1.30 后，打开“开始”菜单→“程序”→“Apache Tomcat4.1”→“Start Tomcat”启动 Tomcat，打开 IE 浏览器，在 IE 浏览器的地址栏中输入 http://localhost:8080/，如果是安装正确，则会出现如图 8-2 所示的画面。

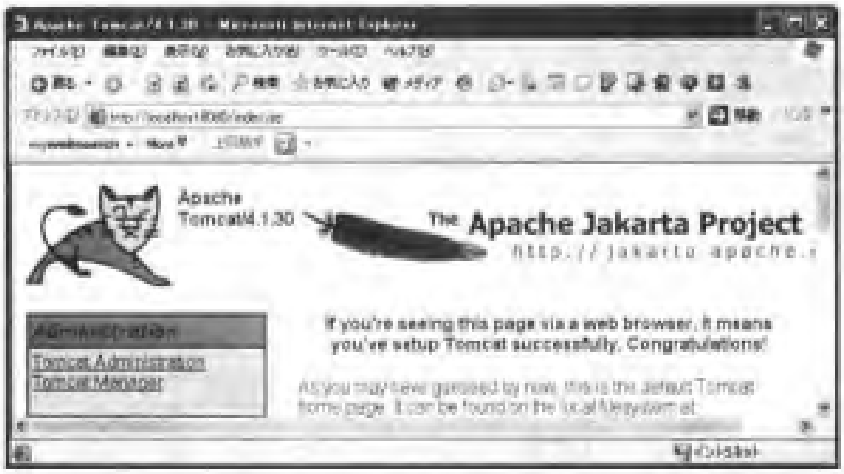


图 8-2　启动 Tomcat 画面

8.2　Tomcat 的配置介绍

8.2.1　Tomcat 目录结构

安装之后的 Tomcat 的目录结构及存放相关文件如表 8-1 所示。

表 8-1　Tomcat 的目录结构及其说明

目 录 名	说　　明
bin	存放启动和关闭 Tomcat 服务的脚本
conf	存放不同的配置文件，包括 server.xml 和 web.xml
work	存放类编译后的 class 文件
webapp	存放应用程序示例及部署文件
logs	存放日志文件
common	存放公用的 jar 文件
server/lib	存放 Tomcat 所需的 jar 文件

8.2.2　server.xml 配置文件介绍

server.xml 配置文件是 Tomcat 的主要配置文件。了解 server.xml 配置文件，不仅有助于我们理解 Tomcat，而且在今后遇到有关修改 server.xml 配置文件时，理解起来更容易些。表 8-2 是 server.xml 文件的一些基本配置说明，具体的配置信息请参考 Tomcat 文档。

表 8-2 Server.xml 文件的基本配置说明

元素名	属 性	解 释	实 例
server	port	指定一个端口，这个端口负责监听关闭 tomcat 的请求	port="8005"
	shutdown	指定向端口发送的命令字符串	shutdown="SHUTDOWN"
service	name	指定 service 的名字	name="Tomcat-Standalone"
connector（表示客户端和 service 之间的连接）	port	指定服务器端要创建的端口号，并在这个端口监听来自客户端的请求	port="8080"
	minProcessors	服务器启动时创建的处理请求的线程数	minProcessors="5"
	maxProcessors	最大可以创建的处理请求的线程数	maxProcessors="75"
	enableLookups	是否可以进行 DNS 查询，如果为 true，则可调用 request.getRemoteHost() 进行 DNS 查询来得到远程客户端的实际主机名，若为 false 则不进行 DNS 查询，而是返回其 IP 地址	enableLookups="true"
	redirectPort	指定服务器正在处理 http 请求时收到了一个 SSL 传输请求后重定向的端口号	redirectPort="8443"
	acceptCount	指定当所有可以使用的处理请求的线程数都被使用时，可以放到处理队列中的请求数，超过该数的请求将不予处理	acceptCount="100"
	connectionTimeout	指定超时的时间（以 ms 为单位）	connectionTimeout="20000"
engine（表示指定 service 中的请求处理机）	defaultHost	指定缺省的处理请求的主机名，它至少与其中的一个 host 元素的 name 属性值是一样的	defaultHost="localhost"
context（表示一个 web 应用程序，通常为 WAR 文件，关于 WAR 的具体信息见 servlet 规范）	docBase	应用程序的路径或者是 WAR 文件存放的路径	docBase="H:\eclipseEnglish\eclipse\workspace\Tomcat_test"
	path	表示此 Web 应用程序的 url 的前缀，表示 http://localhost:8080/path/***	path="/Tomcat_test"
	reloadable	如果为 true，则 Tomcat 会自动检测应用程序的 /WEB-INF/lib 和 /WEB-INF/classes 目录的变化，自动装载新的应用程序，用户可以在不重启 Tomcat 的情况下改变应用程序	reloadable="true"

（续表）

元素名	属　性	解　释	实　例
host（表示一个虚拟主机）	name	指定主机名	name="localhost"
	appBase	存放应用程序的目录	appBase="webapps"
	unpackWARs	如果为 true，则 tomcat 会自动将 WAR 文件解压，否则不解压，直接从 WAR 文件中运行应用程序	unpackWARs="true"
Logger（表示日志、调试和错误信息）	className	指定 logger 使用的类名，此类必须实现 org.apache.catalina.Logger 接口	className="org.apache.catalina.logger.FileLogger"
	prefix	指定 log 文件的前缀	prefix="localhost_log."
	suffix	指定 log 文件的后缀	suffix=".txt"
	timestamp	若为 true，则 log 文件名中要加入时间	timestamp="true"

8.3　Eclipse 与 Tomcat 的连接

8.3.1　Tomcat 插件下载安装

Tomcat 服务器插件是由 Sysdeo 公司开发的，非常小巧，只有 27.8KB。用户可以到网址 http://www.sysdeo.com/eclipse/tomcatPlugin.html 免费下载。写作本书时最新版本为 3.0 beta，是为 Eclipse 3.0 而开发的，这里将下载与 Eclipse 2.1 版本对应的 tomcatPluginV21.zip 文件。另外，这个插件只支持 Tomat 4.0 以上的版本。

将下载的 tomcatPluginV21.zip 文件按原路径解压到 Eclipse 的安装路径下的\plugins 文件夹下，如图 8-3 所示。



图 8-3　Tomcat 服务器插件路径

8.3.2　Tomcat 在 Eclipse 上的配置

安装好插件后需要重新启动 Eclipse。在 Eclipse 上配置 Tomcat 的具体步骤如下所述。

1. 在菜单栏及工具栏中显示 Tomcat

Eclipse 启动后，在菜单栏上的“窗口（W）”菜单选择“定制透视图（Z）”，如图 8-4 所示。

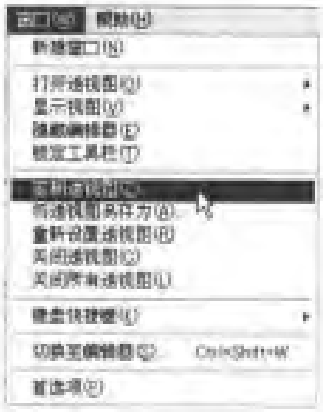


图 8-4 配置 Tomcat

在打开的对话框中选中“其它”，会发现在左边的树型目录中多了一个 Tomcat 选项，选中 Tomcat 的选择框，然后按“确定”按钮，如图 8-5 所示。

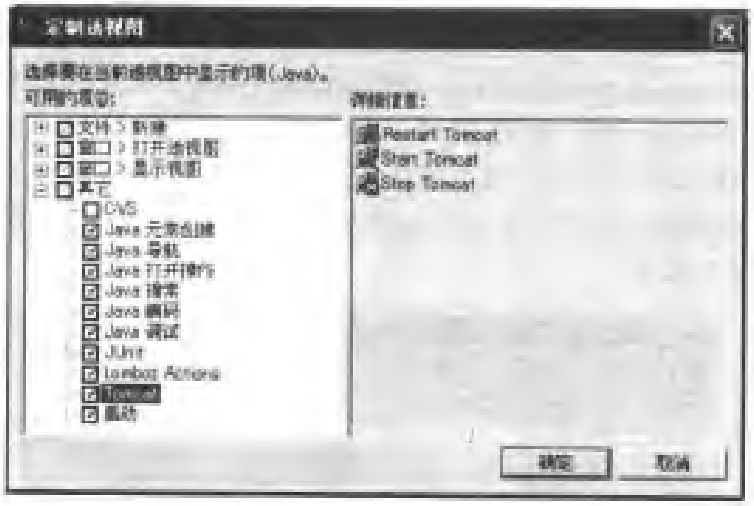


图 8-5 “定制透视图”窗口

之后，你就会发现 Eclipse 上有了两处变化：在菜单栏中多了一个 Tomcat 选项，以及在工具栏中多了三个按钮，如图 8-6 所示。



图 8-6 Tomcat 服务器工具栏按钮

2. 指定 Tomcat 安装根路径

在菜单栏的“窗口（W）”→“首选项（P）”，打开对话框后会发现这里也多了一个 Tomcat 选项，在这里要求选择 Tomcat 的版本，并指定 Tomcat 的安装根路径及 server.xml 文件的路径。本机的 Tomcat 的版本选择 Version 4.1.x，Tomcat 安装根路径为：C:\Tomcat4.1，server.xml 文件的路径会由系统自动找到，如图 8-7 所示。

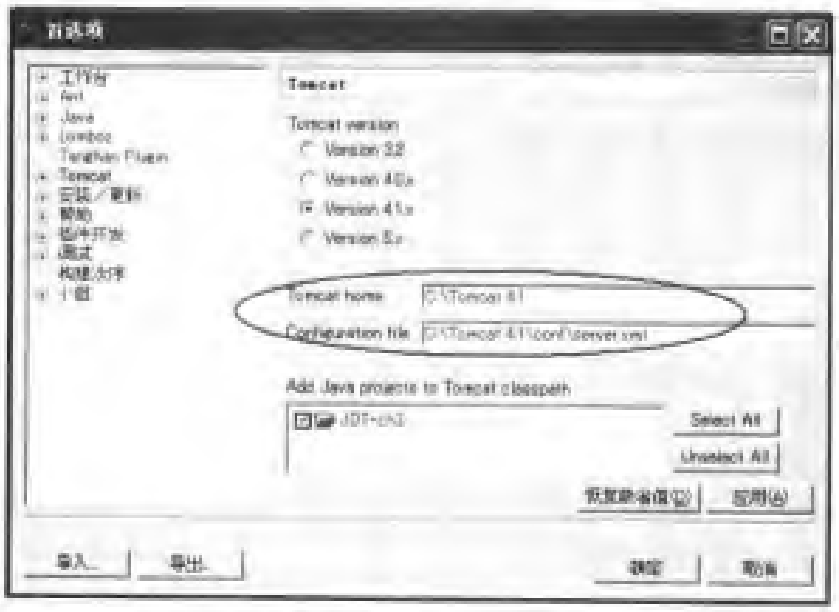


图 8-7　指定 Tomcat 安装根路径

3. 检查指定的 JRE 的位置是否正确

打开“窗口（W）”→“首选项（P）”→“Java”→“已安装的 JRE”中所指定的 JRE 和启动 Tomcat 的 JRE 是否为同一个 JRE，如果不是，可能会导致 Tomcat 不能正常启动，本机中指定的 JRE 的位置在 C:\j2sdk1.4.1_03 下面，如图 8-8 所示。

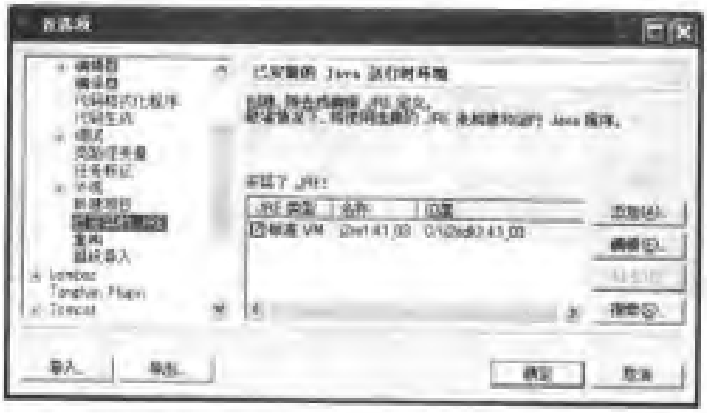


图 8-8　检查指定的 JRE 的位置

正确完成上述步骤后，就可以使用工具栏上的最左边的“小猫”按钮，直接启动 Tomcat 了。要注意的是，启动过程可能会比较慢，要耐心等到在控制台上有以下信息出现：

```
Starting service Tomcat-Standalone
Apache Tomcat/6.1.12
```

之后同样可以在外部浏览器（如 IE）中输入 http://localhost:8080 来测试 Tomcat 是否正常启动。

如果是正常启动，则 Tomcat 与 Eclipse 连接完毕。

8.4 在 Eclipse 平台上开发 Web 应用程序

Web 应用程序有以下不同的组成部分：

- Servlet
- JSP
- JavaBean
- 静态文档（HTML、图像等）
- 部署描述符（web.xml）

本节以一些简单的例子（例 8-1~例 8-3）介绍如何在 Eclipse 平台上开发 Web 应用程序。为了便于读者在初学时容易理解，本例不涉及与数据库的连接（此内容请参阅第 10 章“数据库的构筑与连接”），下面从项目建立、创建 Servlet、部署描述符（web.xml）和创建 JSP 这四个方面进行讲解。

8.4.1 新建 Tomcat 项目

（1）在菜单栏里选择“文件（F）”→“新建（N）”→“项目（R）”，在弹出的对话框中选择“Java”中的“Tomcat Project”项，单击“下一步（N）”，如图 8-9 所示。

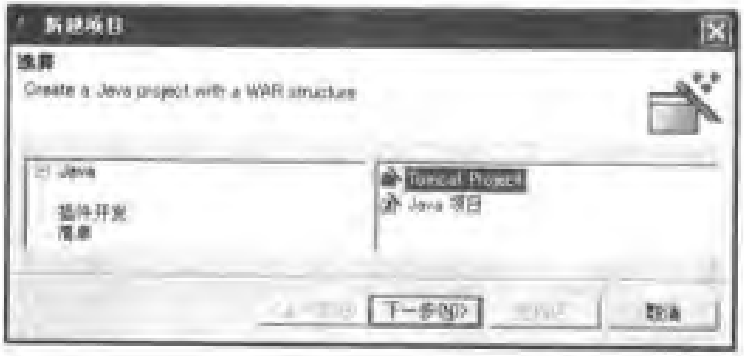


图 8-9 新建 Tomcat 工程

（2）在图 8-10 所示的画面中输入项目名，本章所举例的项目名是：Tomcat_test。如果项目内容选择“使用缺省值（D）”，则该项目自动保存到系统默认目录下 D:\eclipse\workspace\ Tomcat_test 下，反之，则可以由用户自己设定路径。建议使用缺省值，继续按“下一步（N）”。



图 8-10　新建 Tomcat 工程向导

（3）在图 8-11 所示的画面中设定项目的 URI，图中“Context name”文本框中所显示的是系统默认指定的 URI 地址：/Tomcat_test。



图 8-11　选中自动更新 server.xml 文件

注意：必须要选中 server.xml 文件可更新的复选框，如图 8-11 中箭头所指示。这样一来，在新建项目或修改项目名称时，server.xml 文件就可自动更新，而不需要手动修改。

（4）单击“完成”按钮后，则名为 Tomcat_test 的 Tomcat 项目就建成了，在 Eclipse 的导航器中显示了新建的 Tomcat_test 项目，其目录结构如图 8-12 所示。

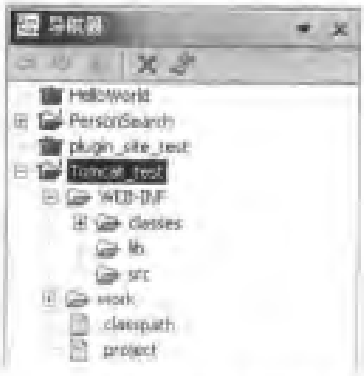


图 8-12　在导航器视图中显示新建的 Tomcat 项目

8.4.2 创建 Tomcat 项目中的 Servlet

Servlet 作为一种 Web 组件，其结构和功能在许多书籍中都作了详细介绍，因此这里就不赘述了。下面介绍在 Eclipse 上做成 Servlet 的步骤和方法。

1. 新建 Servlet 类

在 Tomcat_test 项目上右击鼠标→“新建（W）”→“类（class）”→“下一步（N）”。在如图 8-13 所示的画面中可输入类名和包名。本章所举的 Servlet 的类名为：HelloTomcat，包名为 test。在“超类(S)”一栏中输入 Servlet 所继承的超类：javax.servlet.http.HttpServlet。



图 8-13 新建 Servlet 类

2. 编写 Servlet 源代码

单击“完成（F）”后，则在 Tomcat_test 下的 WEB-INF/src 的文件夹下面生成了名为 HelloTomcat 的 Servlet 类。下面对该类进行代码编辑，你可以采用下面的方法来添加方法：在菜单栏里选择“源（S）”→“覆盖/实现方法（V）”，在弹出的图 8-14 的画面中选择你想要添加的方法，比如 doPost()、doGet()等 Servlet 类中常用的方法。在本例中，选择添加 doGet()方法。

选中方法后，按“确定”，这时你会发现在刚刚生成的 HelloTomcat.java 类中已经生成了 doGet()的方法体，接下来要做的只是在方法体里编写具体要实现的功能代码。以下是例 8-1 的源代码。

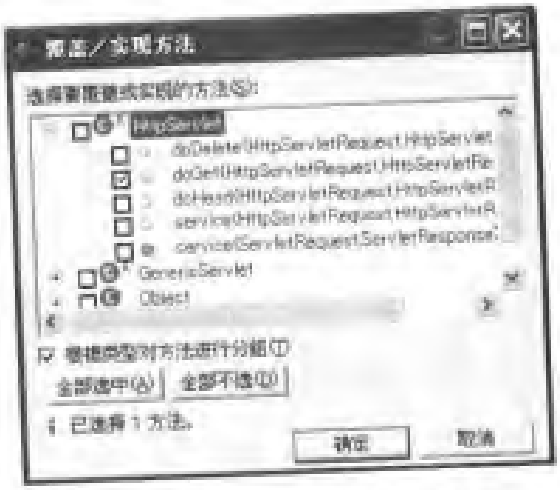


图 8-14　“覆盖/实现方法”窗口

例 8-1：HelloTomcat.java

```
package test;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloTomcat extends HttpServlet {
    protected void doGet(HttpServletRequest arg0, HttpServletResponse arg1)
        throws ServletException, IOException {
        PrintWriter out = arg1.getWriter();
        out.print("Hello Tomcat!");
    }
}
```

这样，Hello Tomcat java 就作好了。但是因为 Web 应用程序还没有被部署，因此还无法在 Tomcat 下面运行这个程序。下面介绍部署描述符 web.xml 的做成。

8.4.3　创建 Tomcat 项目中的部署描述符 web.xml

Web 应用程序是以一组结构化、分层次的文件夹目录形式存在的，在新建的 Tomcat_test 项目下面你可以看到有一个叫 WEB-INF 的特殊文件夹（见图 8-12），在 WEB-INF 文件夹下面有一个名为 web.xml 的部署描述文件，用来给 Servlet 容器提供有关应用程序的配置信息。部署描述文件需要用户根据应用程序的具体情况来手动建立，下面这段代码是本项目的部署描述文件 web.xml 中的代码，从中可以看出是如何把 Servlet 映射给 web.xml 的。

1．生成 web.xml 文件

例 8-2：web.xml 文件

```
<!DOCTYPE web-app
```

```
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <servlet>
        <servlet-name>Test</servlet-name>
        <servlet-class>test.HelloTomcat</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Test</servlet-name>
        <url-pattern>/test/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
</web-app>
```

下面对这个文件进行解释：在这个文件中包含了前面作成的 Servlet 类——HelloTomcat.java。在文件的开头三行中定义了该类的 DTD（文件类型定义），这是在任何 web.xml 文件中必须有的。<servlet>元素在 Servlet 名字与 Servlet 类的全名（包括包名）之间建立了映射，如代码段的第 5 行到第 8 行所作的描述。当 Tomcat 接到请求时，它会判定由哪一个 Servlet 来处理它。<servlet-mapping>元素用来指定处理特定 URL 的 Servlet 类的名字。在这里我们指定以/test 开头的 URL 由 HelloTomcat 来处理，将项目名加在前面，就是以 http://localhost:8080/Tomcat_test/test/开头的 URL 映射为 HelloTomcat。在文件的第 13 行到第 15 行的代码中指定了 Tomcat 引擎会话的超时限制，用<session-config>标签来定义，以 s 为单位。

2. 运行 servlet

将编写好的 web.xml 文件拷贝到该项目的 WEB-INF 文件夹下面，启动 Tomcat。在 IE 的地址栏中输入 http://localhost:8080/Tomcat_test/test/HelloTomcat，会看到显示的结果如图 8-15 所示。



图 8-15 运行 Servlet 结果

8.4.4 创建 Tomcat 项目中的 JSP

1. 创建 JSP 文件

在 Tomcat_test 项目上点鼠标右键→“新建（W）”→“文件”，在弹出对话框里输入文件名，见图 8-16。

本节举例名为 Hello.jsp，在 Tomcat_test/路径下。

注意，不要忘了加后缀名*.jsp。



图 8-16　新建 JSP 文件

之后，你会看见在 Tomcat_test 项目下会多了一个文件 Hello.jsp，如图 8-17 所示。右击鼠标以“文本编辑器”的方式打开该文件，就可以在上面编写 JSP 代码了。在图 8-17 的右半部分，你可以看到本节所举的例子，是一段显示“Hello JSP”的代码。

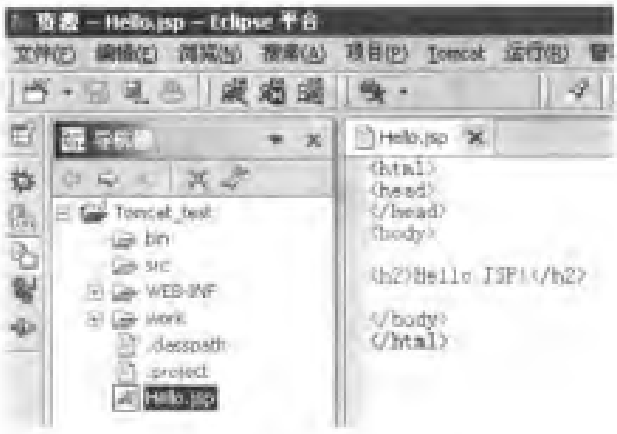


图 8-17　生成 Hello.jsp 文件

2. 运行 JSP

保存文件后，启动 Tomcat（如果 Tomcat 已经启动，则不需要重新启动），在 IE 浏览器的地址栏里输入：http://localhost:8080/Tomcat_test/Hello.jsp，将看到如图 8-18 所示的画面。



图 8-18　运行 Hello.jsp 文件显示的结果

3．被编译成 Servlet 的 jsp 文件——Hello_ jsp.java

运行完 JSP 文件之后，刷新 Tomcat_test 项目，你会在其 work 文件夹下的 jsp 子文件夹下面看到新生成的两个文件，分别是 Hello_jsp.class 和 Hello_jsp.java，如图 8-19 所示。其中 Hello_ jsp.java 文件就是 Tomcat 根据我们的 Hello.jsp 文件创建的 Servlet。读者可以看看 Hello_ jsp.java 的源代码，这样可以帮助你了解 Tomcat 是如何根据 JSP 文件来创建相应 Servlet 代码的。Hello_ jsp.java 的源代码请参见例 8-3：Hello_ jsp.java。



图 8-19　被编译成 servlet 的 jsp 文件

例 8-3：Hello_ jsp.java

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;

public class Hello_jsp extends HttpJspBase {

  private static java.util.Vector _jspx_includes;

  public java.util.List getIncludes() {
    return _jspx_includes;
  }

  public void _jspService(HttpServletRequest request, HttpServletResponse
          response)
```

```
        throws java.io.IOException, ServletException {

    JspFactory _jspxFactory = null;
    javax.servlet.jsp.PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;

    try {
      _jspxFactory = JspFactory.getDefaultFactory();
      response.setContentType("text/html;charset=ISO-8859-1");
      pageContext = _jspxFactory.getPageContext(this, request, response,
                  null, true, 8192, true);
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;

      out.write("<html>\r\n");
      out.write("<head>\r\n");
      out.write("</head>\r\n");
      out.write("<body>\r\n");
      out.write("<h2>Hello JSP!!");
      out.write("</h2>\r\n");
      out.write("</body>\r\n");
      out.write("</html>");
    } catch (Throwable t) {
      out = _jspx_out;
      if (out != null && out.getBufferSize() != 0)
        out.clearBuffer();
      if (pageContext != null) pageContext.handlePageException(t);
    } finally {
  if (_jspxFactory != null) _jspxFactory.releasePageContext(pageContext);
    }
  }
}
```

本章小结

本章通过一组简单例子，讲解了如何利用 Tomcat 插件在 Eclipse 平台上开发 Tomcat 项目，以及创建 servlet，JSP 和 web.xml 文件的方法。可以看出使用 Eclipse 结合 Tomcat 开发 Servlet 应用程序非常方便，而且其功能也能够满足中小型的 Web 应用开发。

第 9 章　使用 Eclipse 进行 J2EE 项目开发

Eclipse 作为一个 Java 应用的 IDE，使用非常方便，但是对于 J2EE 的开发支持还显得不够，本章将向大家推荐一个 Eclipse 的插件来协助 J2EE 开发，这个插件名称叫 Lomboz。Lomboz 插件不仅可用于支持 JSP 语法检查和调试，更主要的作用是能够将很多 Java 应用服务器（如 Tomcat，JBoss，WebLogic 等）、J2EE 组件和 Web 应用开发集成到 Eclipse 中，从而帮助 Java 开发者使用 Eclipse 建立、测试、部署 J2EE 应用，以及 EJB 接口文件和部署文件自动生成等功能。本章将介绍以下内容：

- JBoss 服务器介绍、下载与安装
- Lomboz 插件介绍、下载与配置
- 如何创建 J2EE 项目
- 如何利用 Lomboz 编写和调试 JSP 文件
- 如何利用 Lomboz 生成 EJB

9.1　J2EE Web 应用服务器

9.1.1　介绍几种常用的 J2EE 服务器

提起 J2EE 应用服务器，很自然会联想到两大著名的商业服务器：IBM 的 WebSphere 服务器和 BEA 的 WebLogic 服务器。但是在开发或者学习过程中，有许多免费、轻型并支持 EJB 的应用服务器，例如 JBoss 等。下面将简单介绍各种 J2EE 应用服务器的情况。

1．WebSphere 服务器

WebSphere 服务器以 Java 语言为基础的管理工具应用到了各个方面，包括进行操作管理的管理控制台，进行档案交互应用的应用装配工具，进行逻辑分析的逻辑分析器以及测量和记录运行数据的资源分析工具等。WebSphere 给人印象最深的一点在于，它的逻辑分析器能够通过对在线数据的参考和引用去解释错误信息的能力以及它的资源分析工具所具有的极强的适应性和机动性。

WebSphere 的优点是：

- 支持 J2EE 1.2 标准；
- 详细的、全方位的管理及运行监控工具；
- 友好的中文界面、操作简单；
- 网络服务和 SOAP 支持；
- 插件更新、支持和其他的应用服务器集成。

WebSphere 的不足是：

- 对新 Java 技术的采用滞后；
- 装载灵活性与同类市场竞争产品相比较差。

2. WebLogic 服务器

WebLobic 服务器是美国 BEA 公司开发的一个业界领先的应用服务器。是 Sun J2EE 的纯 Java 实现。WebLogic 服务器提供了对 Sun J2EE 架构的支持。Sun 公司的 J2EE 架构是为企业级提供的一种支持分布式应用的整体框架，为集成后端系统（如 ERP 系统，CRM 系统）以及为实现企业级计算提供了一个简易的、开放的标准。

对于那些正在寻求能够提供 Java 平台所拥有的一切的应用服务器的用户来说，WebLogic 是一个十分正确的选择。但是，它也存在着一些不足，例如它的管理工具比较薄弱，移动性也相对较差，这些都需要提前引起各个公司企业的注意。WebLogic 8.1 的应用服务器管理功能实现延续了 WebLogic 一直坚持的简单、易用的特点，采用基于浏览器的管理界面，这使 WebLogic 应用服务器的配置管理可以通过 Internet 或者 Intranet 来实现远程管理，其他的管理特性同样非常实用。

WebLogic 应用服务器的管理界面完全基于浏览器，所有的管理都可以通过浏览器执行，符合当前用户的使用习惯；也可以编写程序或者通过 Ant 来发布应用。WebLogic 应用服务器的可管理项采用了树形结构来显示，图 9-1 是 WebLogica 英文版的服务器的管理界面。它比较符合人们对 GUI 界面的操作习惯，也使服务器管理项之间的切换和导航非常简单和直观。管理项的属性都采用简单的下拉选择框、文本或者多项选择框等简单控件来表示，使服务器的管理简化为对相关信息的选择、输入，管理者有更多的时间来考虑服务器的相关参数设置，以便服务器能够更加贴近企业应用的实际要求，达到性能的最优化。



图 9-1　WebLogic 服务器的管理界面

此外，WebLogic 应用服务器还具有支持基于文件（.war 或者.ear 文件）或者目录（任意的文件目录）的应用发布、可视化的域配置等优势。WebLogic 应用服务器的不足之处在于运行监控台及管理方面的功能有一定欠缺，保存功能有限。

3．JBoss 介绍

JBoss 是一种遵从 J2EE 规范的开放源代码的 100%纯 Java 的 EJB 服务器，对于 J2EE 有很好的支持。JBoss 核心服务仅是提供 EJB 服务器，不包括 Servlet 和 JSP 的 Web 容器，不过它可以和 Tomcat 完美结合。JBoss 采用 JMX API 实现软件模块的集成与管理。关于 JBoss 的详细资料请参阅其主页 http://www.jboss.org。为了便于读者能够学习，这里选用 JBoss 来作 EJB 服务器。

9.1.2 JBoss 下载与安装

在安装 JBoss 之前，首先确认系统是否已经安装了 JDK。安装 JDK 的步骤可参考第 1 章内容。

JBoss 的下载网址为：http://www.jboss.org。有多种版本可供下载，本机下载的是和 Tomcat 集成一起的 JBoss 版本 jboss-3.0.8_ tomcat-4.1.24.zip。将这个文件解压到 H:\下面。解压缩后，呈现下述目录结构，如图 9-2 所示。

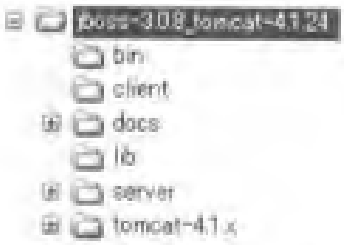


图 9-2 JBOSS 的目录结构

下面介绍各目录下所包含的文件：

- bin——包含可自举的 JAR 文件 run.jar，shutdown.jar 和操作系统相关的启动脚本。run.bat 为 Windows 脚本，run.sh 为 UNIX/Linux 脚本。
- client——包含跟客户端有关的 JAR 文件。
- docs——包含 JBoss/Server 的文档。
- lib——包含自举 JAR 文件。
- server——包含服务器有关的配置文件。
- tomcat-4.1x——包含 Tomcat 的配置文件。

执行 bin/run.bat 文件，在 DOS 界面下出现如下信息：

```
"13:27:54,695 INFO  [Server] JBoss (MX MicroKernel) [3.0.8
(CVSTag=JBoss_3_0_8 Date=200306050849)] Started in 0m:20s:19ms "
```

如果看到这样的信息，就表明 JBoss 已被启动。

注意：在编者下载的 jboss-3.0.8_tomcat-4.1.24 版本里缺少一个 javax.servlet.jar 文件，系统在自动导入 JAR 文件时可能会找不到它，因此可能不会正常启动。解决问题的方法是下载 jboss-3.2.2 版本，解压后在\jboss-3.2.2\server\default\lib 下面，就能找到所需要的 javax.servlet.jar 文件了，将该文件拷贝到 H:\jboss-3.0.8_tomcat-4.1.24\server\default\lib 下面，系统就可以自动找到该文件。

9.2　Lomboz 插件的下载与安装

9.2.1　Lomboz 插件的介绍

Lomboz 插件主要是用来在 Eclipse 上开发 J2EE 的，其主要功能如下：

- HTML 标记和 JSP 关键字的高亮显示、编码援助（code assist）功能、scriptlet 的 Java 编码援助和 JSP 编辑功能。
- JSP 语法检查。
- 利用 Wizard 作成 Web 应用和 EJB 应用。
- 利用 Wizard 新建 EJB 客户端测试程序。
- 利用 XDoclet 生成 EJB 接口、部署描述符的自动生成。
- war 文件和 EJB 档案（JAR）文件的部署。
- 在容器（包括：Tomcat，JBoss，WebLogic）上的调试。

从网站 http://www.objectlearn.com/下载 lomboz.21_02[1].zip 压缩文件，将解压后，将名为 com.objectlearn.jdt.j2ee 的文件夹复制到 Eclipse 的\plugins 下面，如图 9-3 所示。

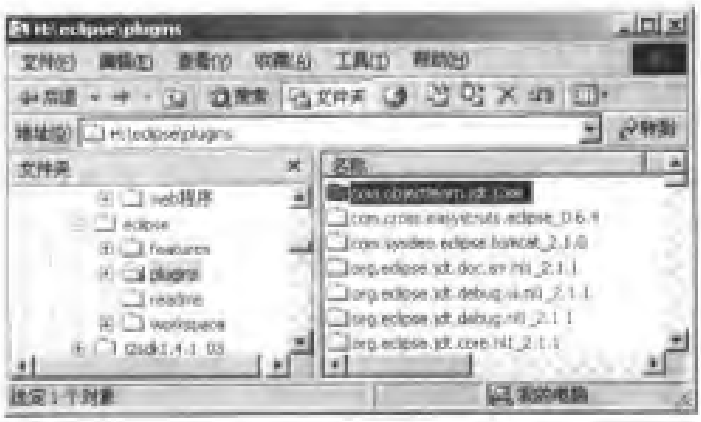


图 9-3　Lomboz 插件安装路径

9.2.2　Lomboz 插件配置

在 Eclipse 上配置 Lomboz 插件可遵循以下步骤：

（1）启动 Eclipse。

（2）进入菜单“窗口（W）”→“首选项（P）”→“Lomboz”选择 J2SDK 的 Tools.jar 的路径，绑定 JDK_TOOLS 的类路径。本机的路径可参见图 9-4。

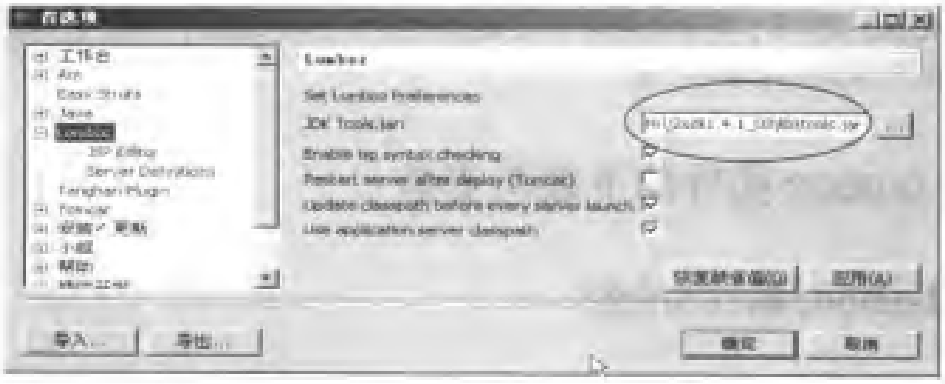


图 9-4　绑定 JDK_TOOLS 的类路径

（3）进入“窗口（W）”→“首选项（P）”→“定制透视图（Z）”，选择 Lomboz J2EE View 保存退出之后在 Eclipse 的工具栏中，如图 9-5 画面中鼠标箭头所指的位置上多了一个按钮。



图 9-5　Lomboz 工具栏按钮

（4）设置新建项目的源文件夹和输出文件夹的指定路径。

为了便于文件的目录管理，可先在新建项目之前指定好项目源文件在缺省情况下使用的路径。打开菜单中的“窗口（W）”→“首选项（P）”→“Java”→“新建项目”，选中“文件夹（F）”，在源文件夹名和输出文件夹名的文本框中输入文件夹名后，单击“应用（A）”，本次使用的系统的默认文件夹为 src 和 bin，如图 9-6 所示。

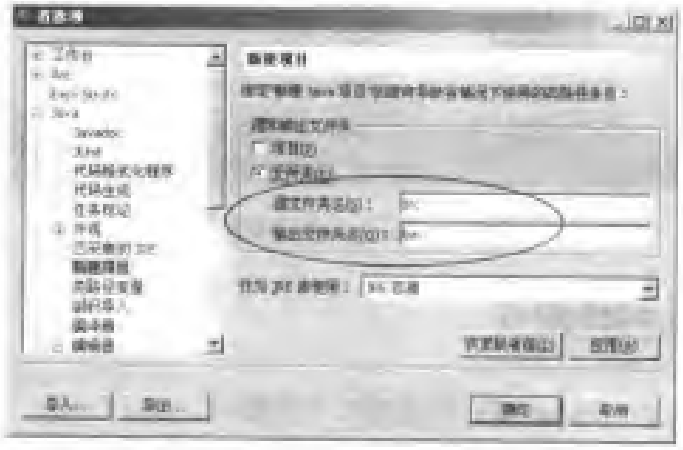


图 9-6　指定项目源文件夹和输出文件夹的路径

（5）设置服务器属性。

执行“窗口（W）”→“首选项（P）”→“Lomboz”→“Server Definitions”，选择服务器类型及设置服务器的路径，图 9-7 是本机所选择的服务器 JBoss 3.0 及其路径。

到此，Lomboz 的基本配置就完成了。

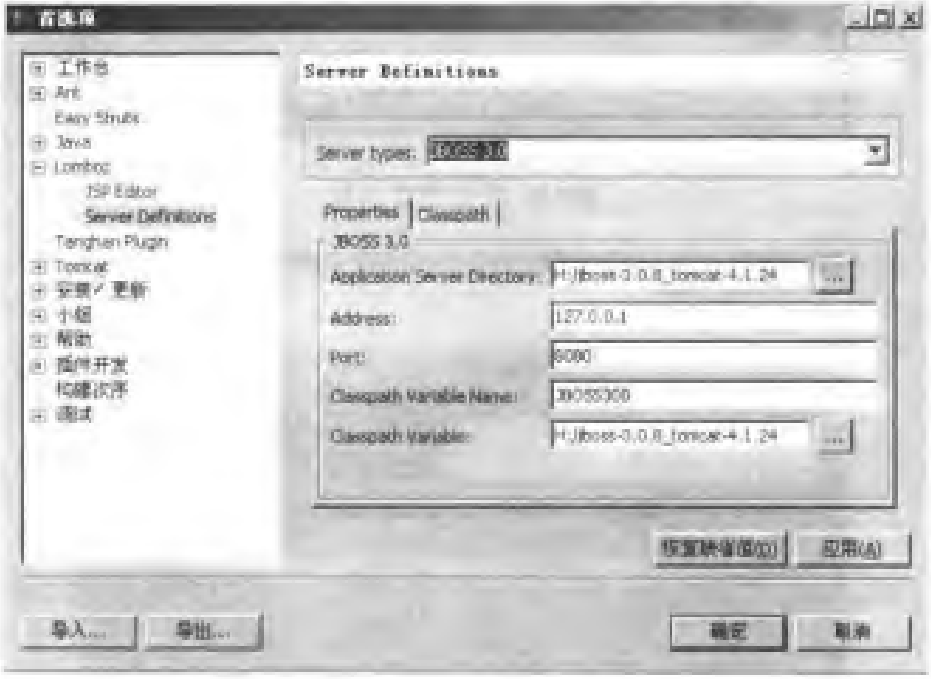


图 9-7　设置服务器属性

注意：每次修改以后，不要忘了单击“应用（A）”。

9.3　创建 J2EE 项目

利用 Lomboz 在 Eclipse 平台上创建 J2EE 项目，可参照以下步骤。

1. 新建 Lomboz J2EE 项目

在菜单栏里选择“文件（F）”→“新建（N）”→“项目（R）”，在弹出的画面图 9-8 中，选择“Lomboz J2EE Project”，然后单击“下一步（N）”。创建一个名为 J2EE_test 的 J2EE 项目，单击“下一步（N）”，默认各种设置，继续按“下一步（N）”，进入下一个对话框，如图 9-9 所示。

2. 添加 Web 模块

在图 9-9 画面中选择“Web Modules”选项卡来添加 Web 模块。在本例中，添加了名为 myWeb 的 Web 模块。为了给以后将要讲到的 EJB 做准备，在这里同时还选择“Ejb Modules”选项卡来添加名为 myEJB 的 EJB 模块，最后选择“Targeted Servers”选项卡来设置目标服务器，本次选择使用的服务器是 JBoss 3.0，如图 9-10 所示。



图 9-8　新建 J2EE 工程



图 9-9　添加 Web 模块

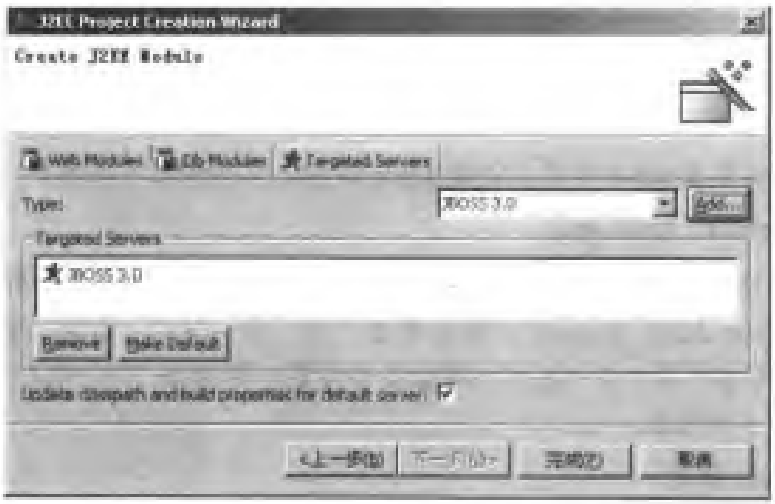


图 9-10　设置目标服务器

完成上述步骤后，单击“完成”，这时就建成了名为 J2EE_test 的 J2EE 项目，该项目的目录层次结构如图 9-11 所示。在 myWeb 模块下自动生成了两个 jsp，分别为 error.jsp 和 index.jsp。



图 9-11　J2EE 工程的目录层次结构

3. 设置 JRE

系统所需的 JRE 文件一般由系统自动加载，也可以由用户自己手动添加。添加 JRE 文件的步骤可以参考图 9-12：选中项目名 J2EE_test 右击鼠标，在其上下文菜单中选择“属性（R）”→“Java 构建路径”→“库（L）”，在这里可以通过选择“添加外部 JAR（X）”或“添加变量（V）”来添加或修改系统所需要的库。



图 9-12　设置 JRE

4. 修改 server.xml 文件

注意：新建 J2EE 项目与新建 Tomcat 项目有一点不同之处，在介绍新建 Tomcat 项目时提到要选择 erver.xml 可以自动修改的选项，而在新建 J2EE 项目时并没有这一选项，因此在启动 Tomcat 之前，需要手动修改 H:/jboss-3.0.8_tomcat-4.1.24/tomcat-4.1.x/conf

/server.xml 文件，修改方法如下所示，在 server.xml 文件的第 380 行附近添加如下代码段：

```
....
</Context>
<Context path="/myWeb" reloadable="true"
    docBase="H:\eclipse\workspace\J2EE_test\myWeb"
    workDir="H:\eclipse\workspace\J2EE_test\j2src" />
</Host>
```

5. 重新指定 Tomcat 服务器的路径

因为这里用到的 Tomcat 服务器与在第 8 章已经设置好的 Tomcat 服务器不是一个，因此不要忘了重新指定 Tomcat 服务器的路径。这里的 J2EE_test 项目指定的服务器是和 Tomcat 集成在一起的 JBoss 3.0 服务器，它所使用的 Tomcat 服务器的路径应该指向新的服务器路径本机的路径是：

```
H:\jboss-3.0.8_tomcat-4.1.24\tomcat-4.1.x
```

如图 9-13 所示，在“Add Java projects to Tomcat classpath”中选中 J2EE_test 项目。这样，J2EE_test 项目的 Tomcat 服务器就设定好了。

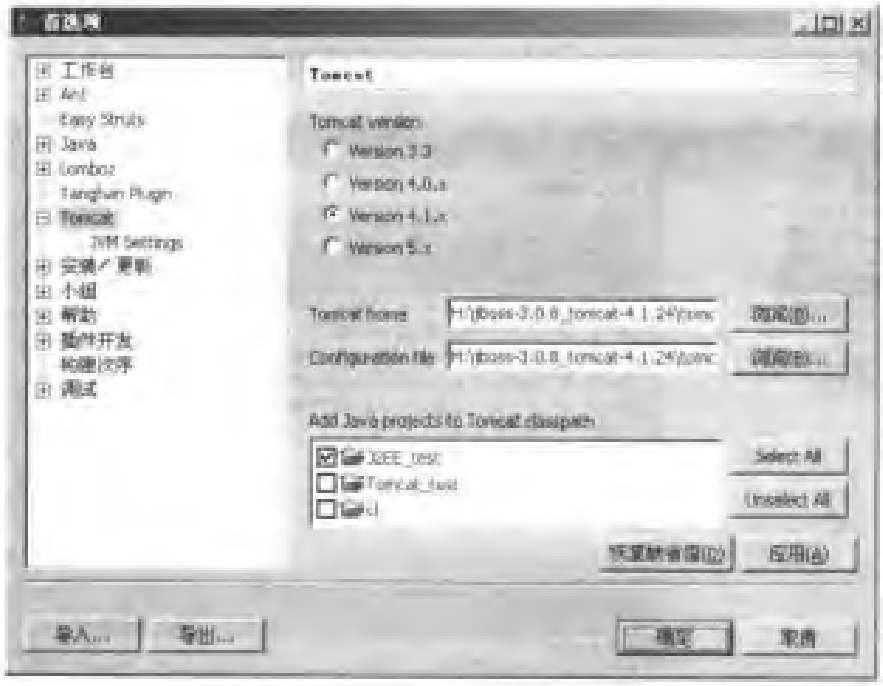


图 9-13　重新指定 Tomcat 服务器的路径

6. 启动 Tomcat

在浏览器的地址栏中输入 http://localhost:8080/myWeb/index.jsp，可看到显示的页面是 index.jsp，如图 9-14 所示。

看到这个显示画面，说明已经正确地创建了一个 J2EE 项目。在这个 J2EE 项目下，同样可以作成 Servlet 类和 JSP 文件，这与第 8 章在 Tomcat 下作成 Servlet 类和 JSP 文件是相同的。惟一不同的是因为 Lomboz 支持 JSP 语法的高亮显示和语法检查，所以在显

示 JSP 文件时会发现与前一章在 Tomcat 项目下显示 JSP 文件时有所不同，下面介绍如何利用 Lomboz 编写和调试 JSP 文件。



图 9-14　启动 Tomcat

9.4　利用 Lomboz 编写和调试 JSP 文件

Lomboz 支持 JSP 语法的高亮显示和语法检查，并且还支持 JSP 调试。下面举例说明如何利用 Lomboz 进行 JSP 文件的编写和调试。这里说明一下，JSP 文件的语法检查和调试功能只能在 Lomboz 项目下进行。本例在刚刚新建的 J2EE_test 项目上来说明。

1. 用 Lomboz JSP 向导来创建 JSP 文件

在 J2EE_test/myWeb 上点鼠标右键，在上下文菜单中选择“新建(W)”→“其它(O)”，在弹出画面图 9-15 中选择“Lomboz JSP Wizard”，用它来生成一个 JSP 文件，然后单击“下一步（N）”，在下一页输入 hello.jsp 文件名，然后单击“完成”，则显示如图 9-16 所示画面。

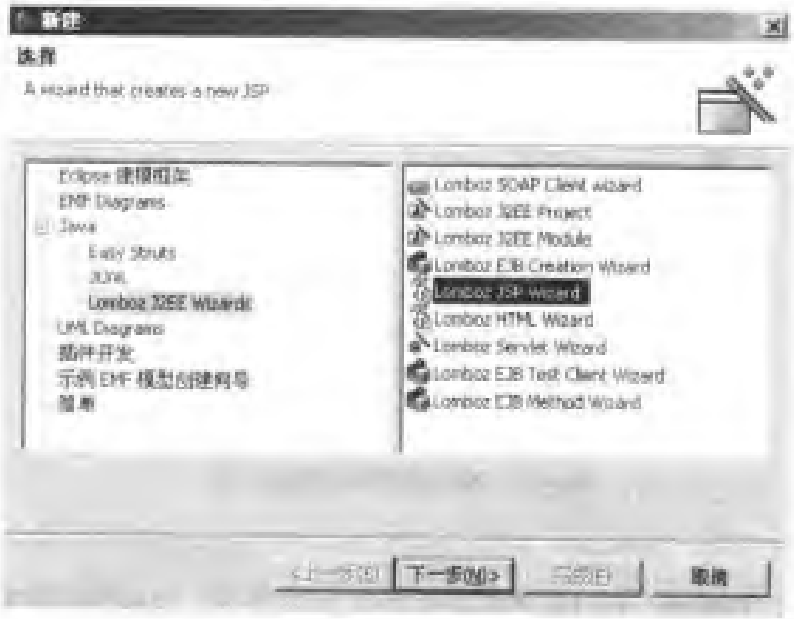


图 9-15　用 Lomboz JSP 向导来创建 JSP 文件

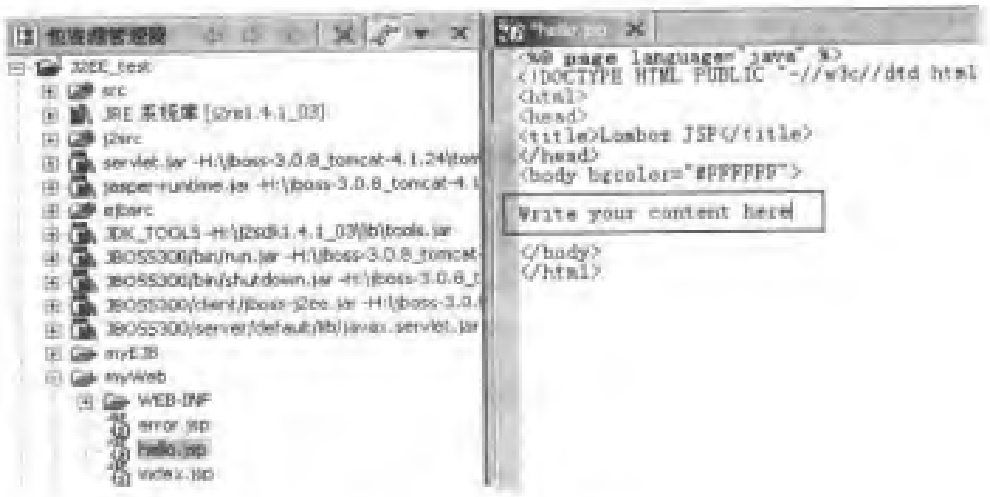


图 9-16　生成 JSP 文件

2. 编写 JSP 文件

在图 9-16 右半边的编辑视图中“Write your content here!”的位置上输入如下代码段：

```
<h2> Hello J2EE! </h2>
<%=new java.util.Date()%>
```

3. JSP 文件的编码援助和高亮显示功能

在编写代码过程中，会发现其不仅有编码援助功能，并且有 HTML 关键字和 Java 语法的高亮显示功能，如图 9-17 所示。在 Tomcat 项目下的 JSP 文件里虽然有 Java 语法的高亮显示功能，但是不具备编码援助功能，只有在“Lomboz J2EE Project”项目下建成的 JSP 文件才具备此功能。



图 9-17　JSP 文件的编码援助功能

4. 启动 IE

保存后启动 Tomcat，在 IE 浏览器的地址栏里输入：http://localhost:8080/myWeb/hello.jsp，将看到如图 9-18 所示的 IE 显示画面。



图 9-18　JSP 文件的运行结果

5. JSP 的语法检查功能

若在代码“Date()”后面加上“;”，这时如果保存文件，可得到如图 9-19 画面上的错误提示，指示在什么地方出现了语法错误，并且在右下方任务栏里提示错误原因。这样就大大改善了以往查找 JSP 错误困难的问题。



图 9-19　JSP 的语法检查功能

以上，介绍了 J2EE 项目的创建和 J2EE 项目下 JSP 文件编译功能的改进。Servlet 的生成与 Tomcat 项目下的 Servlet 的生成相似，在这里就不赘述了。

9.5　Eclipse 结合 Lomboz 开发 EJB

本节将介绍如何在 Eclipse 平台上，利用 Lomboz 来简化 EJB 的开发过程。在 9.2 节介绍 Lomboz 插件时，已经知道 Lomboz 的主要功能除了支持 JSP 编辑外，更有支持简化 EJB 开发的功能，如：利用 Wizard 生成 EJB 源程序文件的雏形、利用 XDoclet 生成 EJB 接口文件和 XML 描述符、EJB 档案（JAR）文件的部署、EJB 容器的调试及利用 Wizard 生成 EJB 测试客户端的雏形等。下面以一个简单的 Stateless Session Bean（无状态 Bean）为例，来说明如何通过利用 Lomboz 其不同的 Wizard 和 Xdoclet 来简化 EJB 繁杂的开发过程。

9.5.1 XDoclet 介绍

XDoclet 是代码生成机，关于它的详细资料可参考 http://xdoclet.sourceforge.net/，XDoclet 从 Rickard Oberg 创建的 EJBDoclet 工具发展而来，它的设想很简单：避免为每个 EJB 提供多个文件，而是从单一 Bean 类文件中提供组件需要的所有信息。通过在 Bean 实现类源文件中插入一些特定的 JavaDoc 标识符“@”，然后让一个 Doclet 工具处理这些标记，由工具为指定的 Bean 生成合适的 XML 描述器文件和接口文件等。

XDoclet 建立在 EJBDoclet 思想的基础上，但适用范围不再局限于 EJB。现在，可以用 XDoclet 生成 Web 服务、Web 应用描述器，甚至还可以对它进行扩展，满足自己的特殊需要。

在 Lomboz 中，由 Wizard 生成包含 XDoclet 所使用的标识符“@”的类。同时，也自动生成了 XDoclet 所要使用的 build.xml 等文件，从而可以直接由 Eclipse 的菜单中直接调出使用 XDoclet。图 9-20 表示在开发过程中 Lomboz 和 XDoclet 关系作用图。

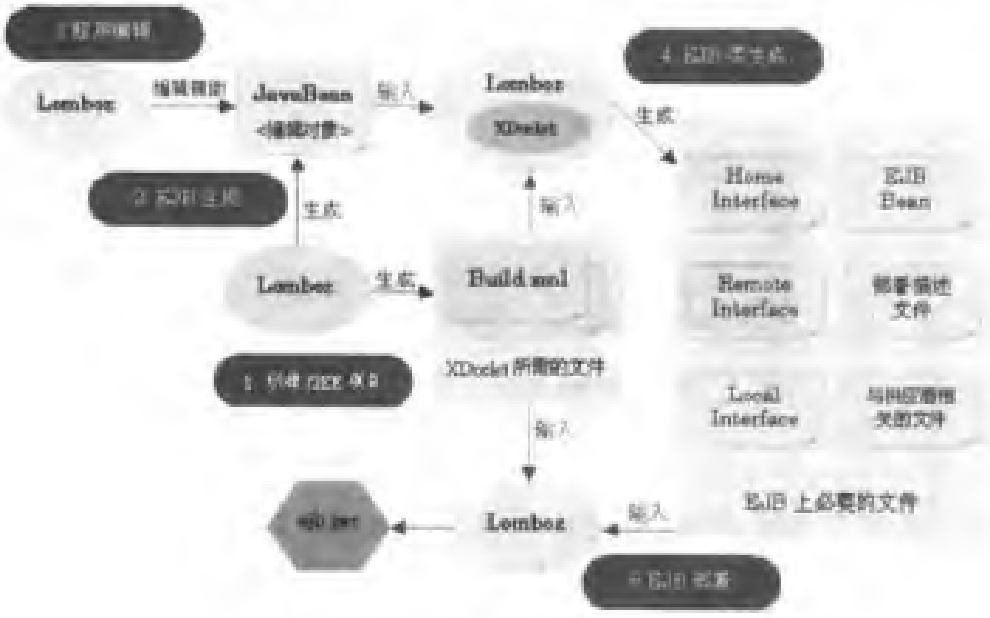


图 9-20 Lomboz 和 XDoclet 关系作用图

9.5.2 生成 EJB

1. 创建 J2EE 项目

创建 J2EE 项目可参考 9.4 节，前面已经新建了一个 J2EE 项目，名叫“J2EE_test”，并且已经添加了名为“myEJB”的 EJB 模块。打开 myEJB/META-INF 文件夹，会看到在它下面已经自动生成了以下文件，见图 9-21 所示。



图 9-21　自动生成的 EJB 文件

2. 生成 EJB

在“J2EE_test”上点右键→“新建（N）”→“Lomboz EJB Creation Wizard”，输入包名和类名，本节例中所起的包名和类名分别是 ml 和 MLAdmin，选择 EJB 类型时，选择的是“Stateless Session Bean”，然后单击“完成”，见图 9-22。



图 9-22　新建无状态 EJB

这样就在 src/ml/文件夹下面新建了一个名为 MLAdminBean.java 的类。其自动生成的代码如下：

```
package ml;

import javax.ejb.SessionBean;

/**
```

```
 * @ejb.bean name="MLAdmin"
 *     jndi-name="MLAdminBean"
 *     type="Stateless"
 **/

public abstract class MLAdminBean implements SessionBean {

}
```

在注释中@ejb.bean name 表示在部署描述符中记述的 EJB 名，该类是 EJB 事务逻辑的实现类。jndi-name 表示在绑定固有的部署描述符记述的 JNDI（Java 命名和目录接口）登录名称。type 表示 EJB 的类型。

3. 源程序的编辑

生成 EJB 后，向该类追加方法，其方法是：右击"MLAdminBean.java"，选择→"Lomboz J2EE..." → "Add EJB method..."，在弹出的对话框里输入完整的方法名，并要选择方法类型和接口类型。"Method Type" 有以下三种方法类型可供选择：

- Business Method——在 EJB 中追加事务方法时选择。
- Home Method——在 Home 接口中追加 Home 方法时选择。
- Create Method——生成 EJB 时调用 ejbCreate，并把它追加到 EJB 中时选择。

接口类型有两种可供选择：远程接口和本地接口，如下所示。

- Remote Interface——远程接口用的方法时选择。
- Local Interface——本地接口用的方法时选择。

本节例子中添加的方法名是：public String getName (String email)。选择方法类型和接口类型分别是 Business Method 和 Remote Interface，见图 9-23。

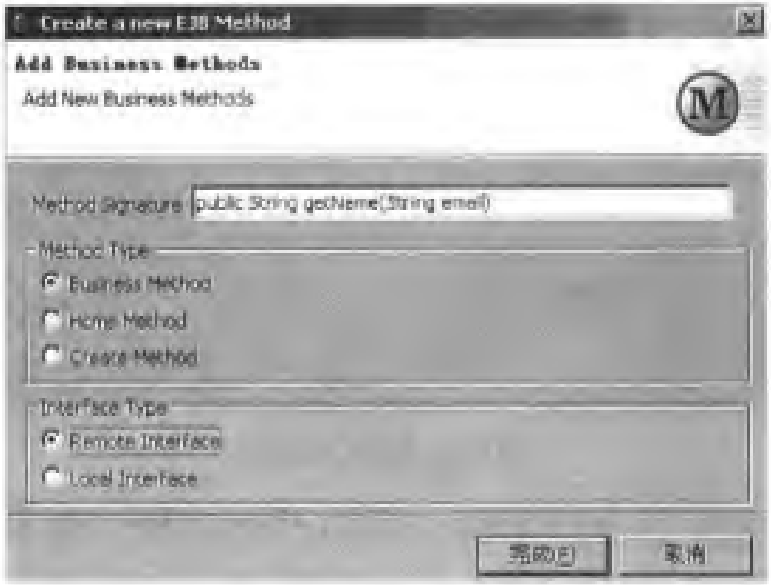


图 9-23 创建 EJB 向导

4. XDoclet 标记

单击"完成"后，在该类中便自动生成了方法的雏形，即如下代码段：

```
/**
*@ejb.interface-method
* tview-type="remote"
**/
public String getName(String email) {
   return null;
}
```

@ejb.interface-method 是 XDoclet 的标记，表明该方法是在 EJB 的远程接口中追加的方法。view-type 参数是从 EJB 2.0 版本以后追加的一个参数，使用追加的本地接口时，该参数才有意义，有以下三种类型的指定：

- Remote——只追加远程接口的方法。
- Local——只追加本地接口的方法。
- Both——同时追加远程/本地接口的方法。

在 Lomboz2.1_02 中，可能会出现下面的错误现象。将参数名指定为“tview-type”，因而错误地将参数生成为了 XDoclet 的标签，这样其结果便与指定默认的 both 的动作没什么区别了。但是，由于在 Lomboz 的 Wizard 中不能指定“both”，并且在远程接口和本地接口中变更方法的事情很少会发生，所以一般不会产生很大的问题。

在方法里编写如下代码，这段代码将实现的功能是，能够获得 email 中"@"前的名称。

```
/** method
* tview-type="remote"
**/
public String getName(String email) {
   return email.split("@")[0];
}
```

9.5.3　生成 EJB 接口类

在 EJB 的开发时，home 接口、远程接口、实现类、部署描述符文件等都是要必须具备的。使用 Lomboz 插件中的 XDoclet，就能从一个 Java 文件自动生成这些文件。下面介绍如何实现。

1. 追加 Java 文件的 EJB 模块

在类 MLAdminBean.java 上右击鼠标→“Lomboz J2EE...”→“Add EJB to module...”。在图 9-24 显示的对话框中，选中“Ejb[myEJB]”，单击“确定”。

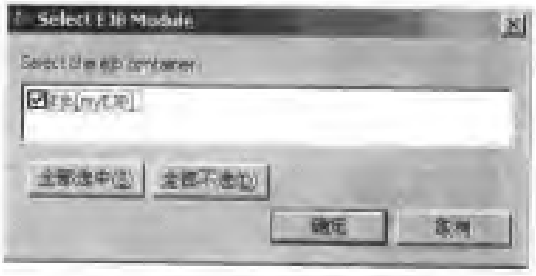


图 9-24　追加 EJB 模块

这时打开beans.xml和ejbs.xml文件，可发现在文件里面已经添加入了MLAdminBean的名称。beans.xml文件的内容如下：

```
<?xml version="1.0" encoding="GBK"?>
<beangroup>
  <bean>
    <beanClass>ml.MLAdminBean</beanClass>
    <type>Session</type>
  </bean>
</beangroup>
```

ejbs.xml文件的内容如下：

```
ml/MLAdminBean.java
```

2. 生成EJB的相关文件

在“myEJB”上右击鼠标，选择→“Lomboz J2EE...”→“Generate EJB Classes”。在ejbsrc文件夹下面自动生成了以下文件（如图9-25所示），并且在myEJB/META-INF文件夹下面的各种服务器部署描述文件也将被自动生成和更新。



图9-25 生成EJB的相关文件

注意：要确保必要的.jar 文件导入到本项目中来。

本项目应包含的.jar 文件如图 9-26 所示。

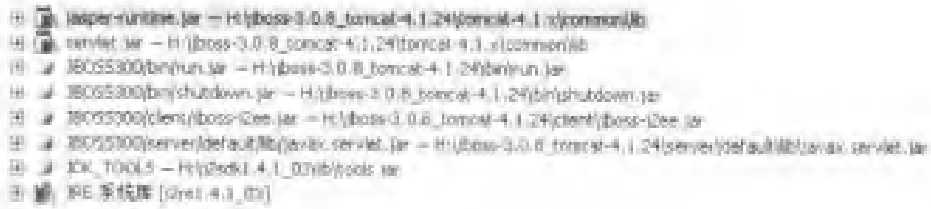


图 9-26　项目应包含的.jar 文件

自动生成的部署描述符（ejb-jar.xml）的源代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
        2.0//EN"
"http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar >
  <description><![CDATA[No Description.]]></description>
  <display-name>Generated by XDoclet</display-name>
  <enterprise-beans>

    <!-- Session Beans -->
    <session >
      <description><![CDATA[ ]]></description>
      <ejb-name>MLAdmin</ejb-name>
      <home>ml.MLAdminHome</home>
      <remote>ml.MLAdmin</remote>
      <local-home>ml.MLAdminLocalHome</local-home>
      <local>ml.MLAdminLocal</local>
      <ejb-class>ml.MLAdminBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>

  </enterprise-beans>
  <assembly-descriptor >
  </assembly-descriptor>
</ejb-jar>
```

9.5.4　EJB 部署

部署时将所必需的文件作为 EJB 的 jar 文件被制成文档，然后被拷贝到应用程序服务器的相应路径下。在 Eclipse 中这个过程非常简单，只需在“myEJB”文件夹右击鼠标→“Lomboz J2EE...”→“Deploy Module”，在“myEJB”文件夹下面就生成了 myEJB.jar 文件，如图 9-27 所示。

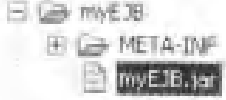


图 9-27　部署 myEJB.jar 文件

并且，在 myEJB/META-INF/build.properties 文件里的 deploy.dir 属性也跟着自动编辑更新。

下面是本例中 build.properties 文件的源代码：

```
container=myEJB
deploy.dir=H:/jboss-3.0.8_tomcat-4.1.24/server/default/deploy
eclipse.home=H:/eclipse/
```

9.5.5 生成客户端 EJB

为了测试以上生成的 EJB 是否正确可行，现在利用 Lomboz 插件功能来生成客户端 EJB 的测试程序。

1. 创建客户端项目

选择“文件”→“新建”→“项目”→“Java”→“Java 项目”，在弹出的对话框中输入项目名，本节举例的客户端测试的项目名为“ClientTest”。单击“下一步”，在图 9-28 中，在“项目（P）”中的“构建路径上所必需的项目（R）”栏中，选中“J2EE_test”的复选框，然后单击“完成”。



图 9-28 客户端项目的生成

2. 在建成的项目中新建客户端的测试程序

在“ClientTest”上右击鼠标→“其它（O）”→“Lomboz J2EE Project”→“Lomboz EJB Test Client Wizard”，弹出的对话框如图 9-29，在对话框中输入。

单击“完成（F）”后，在 Client/src/mltest/下面得到 MLTest.java 的类。EJB 客户端所用的库大多数是在新建项目时由系统自动追加而成的。不过，用户也可能会遇到图 9-30 中的任务栏中所显示的错误。这是由于系统没有在 H:/jboss-3.0.8_tomcat-4.1.24/client 下面找到 jaas.jar 文件。

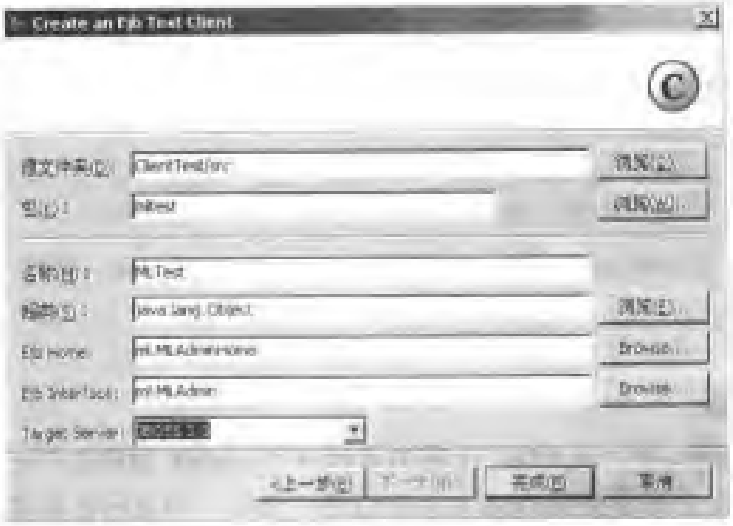


图 9-29　新建测试程序

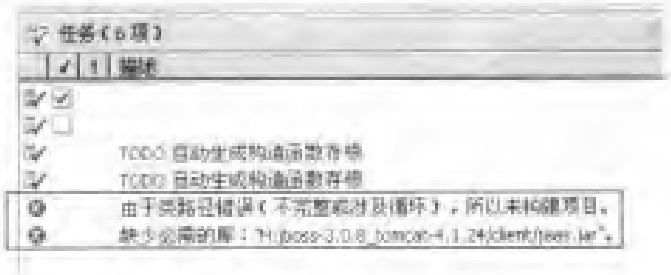


图 9-30　缺少 jar 文件的错误提示

修改这一错误的方法是：在“ClientTest”项目上面单击右键→“属性（R）”→“Java 构建路径”，先删除这个错误的构建路径，然后再添加 jaas.jar 文件的正确路径，本机 jaas.jar 文件的正确路径是 H:\Tomcat 4.1\server\lib。添加之后会发现该错误自动消失了。图 9-31 是“ClientTest”项目下所必须的库文件。

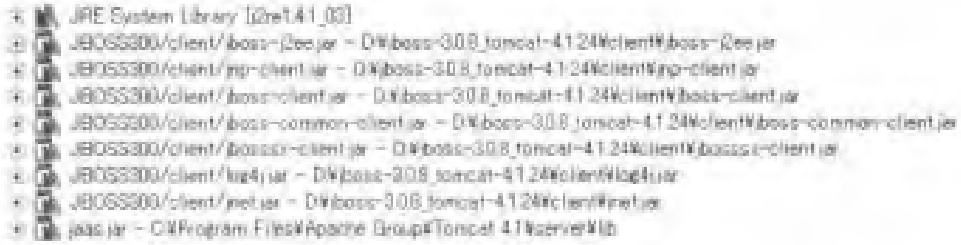


图 9-31　“ClientTest”项目下所必须的库文件

9.5.6　运行 JBoss 服务器

可以参照如下代码来修改程序代码。保存修改后运行 JBoss 服务器，查看程序运行结果。

```
public void testBean() {

    try {
        ml.MLAdmin myBean = getHome().create();

//--------------------------------------
        //This is the place you make your calls.
        //System.out.println(myBean.callYourMethod());

        System.out.println(myBean.getName("DKD@sohu.com"));

    } catch (RemoteException e) {
    ...
```

1. 运行服务器

代码修改保存后，在“myEJB”上右击鼠标→“Lomboz J2EE...”→“Debug Server”，在 Eclipse 的控制台上将看到 JBOSS 启动的信息，最后一句显示信息如下：

```
    14:17:11,640 INFO  [Server] JBoss (MX MicroKernel) [3.0.8 (CVSTag=JBoss_3_0_8
Date=200306050849)] Started in 0m:9s:328ms
```

2. 运行测试程序

选中 MLTest.java 类后，单击“运行（R）”，运行方式选择“Java 应用程序”，在控制台上可以看到如下结果：

```
DKD
```

本章小结

本章讲解了 JBoss 服务器及其下载与安装、Lomboz 插件的介绍及配置，并且通过一个实例详细介绍了如何创建一个 J2EE 项目，并在 J2EE 项目下利用 Lomboz 插件来调试 JSP，以及利用 Lomboz 插件来简化 EJB 的开发过程。

本章重点在于介绍如何利用 Lomboz 插件在 Eclipse 平台上开发 J2EE 项目，因此举的相关例子是最基本的应用，读者若想了解更多有关 J2EE 和 EJB 方面的知识，请参考《精通 EJB（第二版）》（Ed Roman 著，电子工业出版社）等这方面的专业书籍。

第六篇

Eclipse 数据库应用

PART 6

第 10 章　数据库的构筑与连接

第 10 章　数据库的构筑与连接

前两章中有关 Web 应用程序和 J2EE 应用程序的例子没有涉及到与数据库的连接问题，但是真正应用到实际的应用程序都离不开数据库连接，从而进行检索、更新、插入或删除数据库里的数据。Eclipse 平台上的数据库连接采用的是 JDBC 连接数据库的方式。本章将介绍以下内容：

- MySQL 数据库构建
- JDBC 对象
- Tanghan Plugin 的使用
- EJB 与数据库的连接

10.1　数据库的构筑

10.1.1　MySQL 数据库简介

MySQL 数据库是众多的关系型数据库产品中的一个，MySQL 是以操作简便著称的开放源代码的关系型数据库管理系统（简称 RDBMS），它相比较其他系统而言，具有运行速度快、多线程、多用户等特点，并且可以在 UNIX\Linux\Windows 等多个平台上进行操作。除了具有许多其他数据库所不具备的功能和选择外，MySQL 数据库是一种完全免费的产品，用户可以直接从网上下载数据库，用于个人或商业用途，不必支付任何费用。

MySQL 的主要性能如下：

- 提供 C，C++，Java(JDBC)，Perl，Python，PHP 和 TCL 的 API 接口。
- 完全多线程，适于多 CPU 使用。
- 多平台，包括 Solaris，SunOS，BSDI，SGI IRIX，AIX，DEC UNIX，Linux，FreeBSD，CO OpenServer，NetBSD，OpenBSD，HPUX，Win9x and NT。
- 数据类型多样，包括 integers，float，double，char，varchar，text，blob，date，datetime，year，set，enum。
- 非常灵活和安全的权限系统，密码加密。
- 可处理大型数据（超过 5000 万条记录）。
- 经权威商业内存泄露软件检测无内存泄露。
- 多种语言支持（但是不支持中文）。

在大概了解了 MySQL 之后，下面讲解 MySQL 的安装方法。

10.1.2 MySQL 数据库安装

这里主要介绍在 Windows 环境下安装 MySQL。可以从下面的 URL 中下载得到 MySQL：http://www.mysql.com/downloads/index.html。

目前，公开发布的最新版本是 5.0 版本，在本次的介绍中使用的是官方网站推荐使用的 4.0 版本的 Windows 版本 mysql-4.0.16b-win.zip。将下载所得到的压缩文件解压，然后运行文件夹内的 SETUP.EXE 文件，可启动安装程序，按照窗口提示进行安装。

现将 MySQL 安装在 H:\mysql 路径下。安装完成后，运行 H:\mysql\bin 下的 winmysqladmin.exe 文件，可看到如图 10-1 所示的画面，在文本框中输入用户名和密码，本机输入的用户名和密码都是 admin。

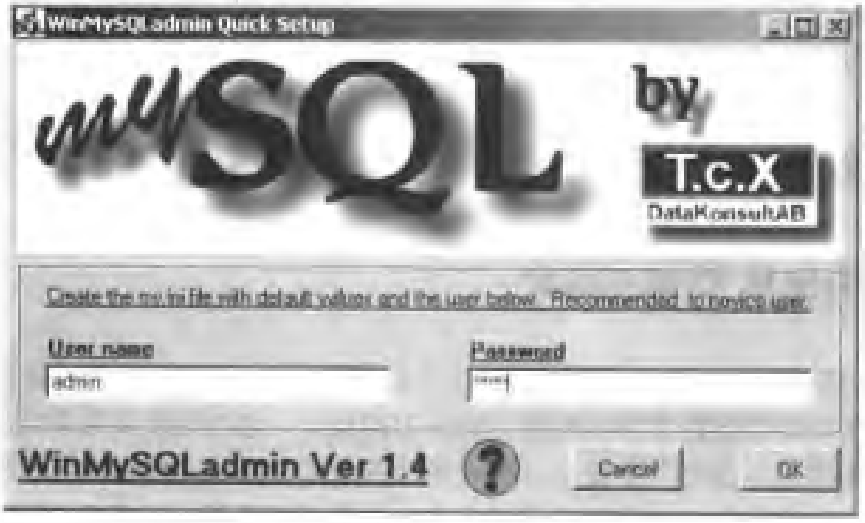


图 10-1 MySQL 启动画面

按“OK”按钮后，会在 C:\Windows 下自动生成 my.ini 文件。并且，再运行 winmysqladmin.exe 时会出现 mysql 的主菜单画面。选择“my.ini Setup”选项卡，可以看到生成的 my.ini 文件的内容，如图 10-2 所示，并且可以进行修改。

10.1.3 MySQL 数据库基本操作指令

1. 数据库的做成

首先做成一个管理邮件列表的数据库“mldb”。在做成数据库的时候，使用 mysqladmin 的 create 命令。在 MySQL 安装路径下的\bin 下面，本机的路径是 H:\mysql\bin，实行下面的命令：“mysqladmin create mldb”。

如果用户想确认一下数据库的做成是否成功，请使用 mysqlshow 命令，参考图 10-3。



图 10-2　mysql 的主菜单画面

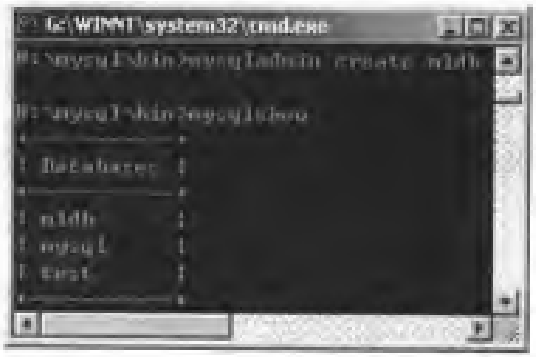


图 10-3　MySQL 数据库的作成

2. 建立用户

接下来，建立一个向 MySQL 连接的 mailAdmin 用户。

（1）用以下的命令，登录到 MySQL 的根目录下。

```
mysql -u root
```

（2）运行以下的命令语句，建立与 mldb 相关的全权用户，本机用户名为 mailAdmin。

```
GRANT ALL ON mldb.* TO mailAdmin@localhost IDENTIFIED BY 'mailAdmin';
GRANT ALL ON mldb.* TO mailAdmin@'%' IDENTIFIED BY 'mailAdmin';
```

第一条语句规定的是向本地终端连接的权限，第二条语句规定的是向其他终端连接的权限。如果想要确认一下该用户有什么样的使用权限，请使用以下的命令。

```
select user, host, password from mysql.user;
```

注意：显示出来的密码是已被加密了的。

3. 表的建成

先来建一个管理邮件地址和姓名的表，表名叫“mlmember”。

（1）先以 mailAdmin 用户身份登录。

```
H:\mysql\bin\mysql -u mailAdmin -pmailAdmin mldb
```

（2）执行建表的 DDL。

```
create table mlmember(
    email varchar(100),
    name varchar(100),
      primary key(email)
);
```

图 10-4 是运行上述两条命令的界面，并且可以从图 10-5 的 MySQL 图形管理界面上看到命令执行的结果。



图 10-4　运行 MySQL 命令的界面

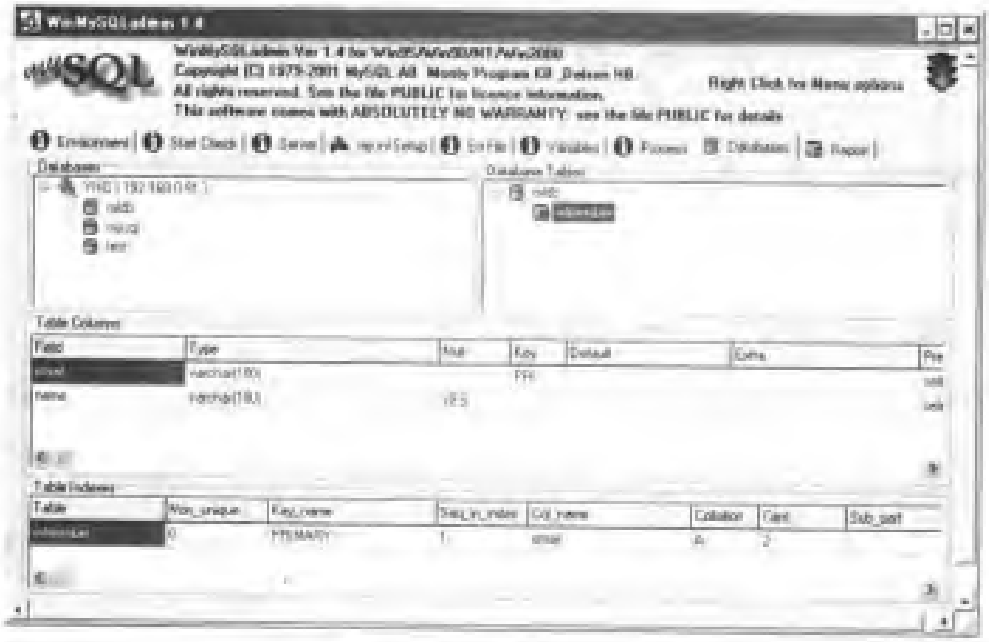


图 10-5　MySQL 图形管理界面

4. 插入数据

现在执行下面的语句可以实现向表 mlmember 里插入一条数据。

```
insert into mlmember values("Mike@sohu.com", "Mike");
```

5. 查询数据

用户可以使用 select 命令进行数据的查询，例如：通过下面的语句来查询名字为“Mike”的所有记录。

```
select * from mlmember where name = "Mike";
```

6. 删除和修改数据

用户可以使用 delete 命令来删除表中已有的记录。例如：通过下面的语句就可以把表中名字为“Mike”的所有记录删除掉。

```
delete from mlmember where name = "Mike";
```

用户还可以使用 update 命令来对表中的数据进行修改。例如：通过下面的语句来修改名字为“Mike”的 Email 地址。

```
update mlmember set email="Mike@hotmail.com " where name = "Mike";
```

以上只简单介绍了基本的 Mysql 命令，如果需要详细了解 MySQL 数据库管理系统，请参考专门介绍 MySQL 的有关资料。

10.2　数据库连接

10.2.1　JDBC 对象

1. JDBC 概念

JDBC 接口是一种纯粹的 Java API。JDBC API 是 Java 程序语言的应用程序接口，它提供数据访问的基本功能。利用 JDBC API 就可以使用同一语法访问不同的关系型数据库，方便于使用不同关系型数据库的不同应用程序的组件，体现了 Java 的平台无关性。

2. JDBC 的用途

JDBC 提供一组类和接口，供开发人员编写数据库应用。基本的 JDBC 交互操作分四个步骤：

（1）打开数据库连接；

（2）执行 SQL 语句；

（3）处理返回的结果；

（4）关闭数据库连接。

下列的代码段分别表示了上述的四个步骤：

```
//step1: 加载 postgresql 驱动程序
Class.forName("org.postgresql.Driver");
//建立与数据库的连接
con=DriverManager.getConnection(jdbc:postgresql://localhost/kaihi03?user
    = postgres&password=hogehoge);
//step2: 执行 SQL 语句
Statement st = con.createStatement();
ResultSet rs = st.executeQuery(sql);

//step3: 处理返回的结果
while (rs.next()) {
    out.println("ID="+ rs.getInt("type_id" ));
}

rs.close();
st.close();
//step4: 关闭数据库连接
con.close();
```

10.2.2 JDBC 驱动类型

JDBC 的数据访问借助驱动程序来完成，驱动程序将实现 java.sql.Driver 接口。通常，驱动程序可以被认为是 Java 应用程序和后台数据库之间的中间层，它从应用程序中接收 SQL 命令，并将它们导入数据库中，然后获取结果，并将结果返回调用程序。

JDBC 驱动程序类型共分四种：JDBC-ODBC 桥、自带 API 的部分 Java 驱动程序、JDBC-Net 纯 Java 驱动程序、本地协议 Java 驱动程序。下面就一一介绍这几种驱动类型的具体情况。

1. JDBC-ODBC 桥驱动程序类型

这种类型的驱动程序通常适用于公司网络，或者是 DBMS 本身不支持 JDBC 的环境。它通过 ODBC 驱动程序提供 JDBC 数据库访问支持，因此客户机必须配置 ODBC 驱动程序，以便 JDBC-ODBC 桥能够工作。

2. 自带 API 的部分 Java 驱动程序

这种类型的驱动程序一部分采用 Java 编写，一部分采用本地代码，它把客户机上的 JDBC 调用转换为数据库管理系统（例如 Oracle 等）的调用。该驱动程序依赖于它所连接的特定数据源的客户端库文件，同样需要依赖于本地库文件，因此不易于移植。

3. JDBC-Net 纯 Java 驱动程序类型

这种类型的驱动程序把客户端的 JDBC 调用转换成独立于数据库的网络协议，然后将请求发送给自定义的中间件服务器，该服务器会把网络协议转换成 DBMS 特定的协议，并发给数据库，执行结果再循原路由经中间件服务器返回给客户端。这样一来，中间件服务器就可以将 Java 客户端与众多不同的数据库相连，该驱动程序使用灵活，非常适用于基于网络的应用程序。

4. 本地协议 Java 驱动程序

这种类型的驱动程序将 JDBC 调用直接转换为 DBMS 所使用的网络协议，它把 JDBC

命令直接转换成数据库引擎使用的网络协议，允许从客户机机器上直接调用 DBMS 服务器，是 Intranet 访问的一个很实用的解决方法。与其他三种驱动程序相比，该类型的驱动程序有非常明显的优势，即它没有额外的翻译和中间层，这将很大程度地改善系统的性能。

10.2.3　JDBC 的使用过程

使用 JDBC 需要完成以下步骤。

1．加载 JDBC 驱动程序

将驱动程序类加载到应用程序的 JVM 中，使用驱动程序可以打开 DB 连接。另外，可以在应用程序中使用 Class.forName()方法加载 JDBC 驱动程序，并且要确保用户的 JDBC 驱动程序位于服务器中的 CLASS-PATH 中,或将 JDBC 驱动程序拷贝到 Web 应用程序的 WEB-INF/lib 目录下。

以下是几种常用数据库加载 JDBC 驱动程序的方法：

```
Class.forName( "com.mysql.jdbc.Driver" ); // MySql 的加载 JDBC 驱动程序的方法
Class.forName( "org.postgresql.Driver" );
                              // postgresql 的加载 JDBC 驱动程序的方法
Class.forName( "oracle.jdbc.driver.OracleDriver" );
                              // oracle 的加载 JDBC 驱动程序的方法
Class.forName( "com.sybase.jdbc2.jdbc.SybDriver" );
                              // sybase 的加载 JDBC 驱动程序的方法
Class.forName( "com.microsoft.jdbc.sqlserver.SQLServerDriver" );
                              // SQLServer 的加载 JDBC 驱动程序的方法
Class.forName("com.ibm.db2.jdbc.net.DB2Driver");
                              // DB2 的加载 JDBC 驱动程序的方法
```

2．建立数据库连接

java.sql.DriverManager 类负责管理驱动程序信息，一旦载入了驱动程序，就被该类识别为可以用的驱动程序。用 DriverManager.getConnection()方法连接到 DBMS，该方法接收一个数据库的 URL 作为参数，如果 DBMS 需要的话，参数还可以包括用户 ID 和口令。该方法返回一个用于整个处理过程的 Connection 接口，程序通过它引用数据库。以下是与几种常用数据库的方法。

（1）与 MySQL 数据库建立连接的方法：

```
Connnection con =
                DriverManager.getConnection("jdbc:mysql://MyDbComputerNa
                meOrIP:3306/myDBName", ID, Pwd );
```

（2）与 postgresql 数据库建立连接的方法：

```
Connnection con =
                DriverManager.getConnection("jdbc:postgresql://MyDbCompu
                terNameOrIP:3306/myDBName", ID, Pwd );
```

（3）与 Oracle 数据库建立连接的方法：

```
Connnection con =
                DriverManager.getConnection("jdbc:oracle:thin:@MyDbCompu
```

```
terNameOrIP:1521:ORCL ", ID, Pwd );
```

（4）与 Sybase 数据库建立连接的方法：

```
Connnection con =
                DriverManager.getConnection( "jdbc:sybase:Tds:MyDbComput
                erNameOrIP:2638 ", ID, Pwd);
```

（5）与 SQL Server 数据库建立连接的方法：

```
Connnection con =
                DriverManager.getConnection("jdbc:microsoft:sqlserver://
                MyDbComputerNameOrIP:1433:DatabaseName=master", ID, Pwd);
```

（6）与 DB2 数据库建立连接的方法：

```
Connnection con =
                DriverManager.getConnection("jdbc:db2://
                MyDbComputerNameOrIP/ myDBName ", ID, Pwd );
```

3. 创建并执行 SQL 语句

用 Connection.createStatement()方法来创建一个 Statement 对象。Statement 对象执行一个查询并从 DBMS 返回一个包含响应结果的 ResultSet（结果集）对象。

4. 处理从 DBMS 返回的数据

查询处理后从 DBMS 收到的查询结果赋给 java.sql.ResultSet 对象。该对象包含一些用来从结果集中获取数据存到 Java 变量中的方法，以便进行进一步处理。

5. 关闭数据库连接

使用与数据库有关的对象所消耗的内存是很大的，因此在完成数据库访问后要及时关闭数据库连接，同时还应关闭 Connection 中返回的内容，包括 Statement 对象和 ResultSet 对象。

以上是数据库的连接、数据库查询及数据库断开的全部过程。下面通过一个简单的例子说明上述的这些过程，帮助读者有一个更直观地理解。

10.2.4 一个简单的数据库查询的例子

下面是一个最基本的数据库连接的例子，它包含了使用 JDBC 的一个完整过程，请参考下面例程的注释。该例子表示如果数据库连接成功，则在控制台上显示出“DB connection success，your e_mail is：从数据库里选出的邮箱地址”。dbConn.java 类的源代码如下。

例 10-1：dbConn.java

```
import java.sql.*;

public class dbConn {
  public dbConn() {
    super();
  }
```

```
public static void main(String[ ] args) {
  try {
    Connection con;
    //加载数据库驱动程序
    Class.forName("com.mysql.jdbc.Driver");
    //连接数据库

    con = DriverManager.getConnection("jdbc:mysql://localhost/mldb");

    //创建一个 Statement 对象
    Statement st = con.createStatement();
    // sql 查询语句
    String sql = "";
    sql = "select * from mlmember where name='Mike';";

    //创建一个 ResultSet 对象
    ResultSet rs = st.executeQuery(sql);
    if(rs.next()) {
      //处理从 DBMS 返回的数据
      String s_email = rs.getString(1);
      System.out.println("DB connection success your e_mail is : "+ s_email);
    }
    //关闭 ResultSet 对象
    if(rs != null) rs.close();
    //关闭 Statement 对象
    if(st != null) st.close();
    //关闭数据库连接
    if(con != null) con.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

注意：在运行该例子前必须把 MySQL 驱动程序导入本例的项目中。

10.3　Tanghan 插件

10.3.1　Tanghan 插件简介

Tanghan 插件主要是为 Eclipse 开发的，用来通过标准的 JDBC 接口访问数据库。

1．Tanghan 插件组成

Tanghan 插件主要由两个视图和一个编辑器组成。视图包括显示数据库连接和驱动的视图及显示表数据的视图，分别如图 10-6、图 10-7 所示。显示数据库连接和驱动的视图主要用于记录有哪些数据库驱动及与数据库的连接。

显示表数据的视图主要用来显示数据库中表的字段，还可以在该视图中运行简单的 SQL 语句。



图 10-6　数据库连接和驱动的视图

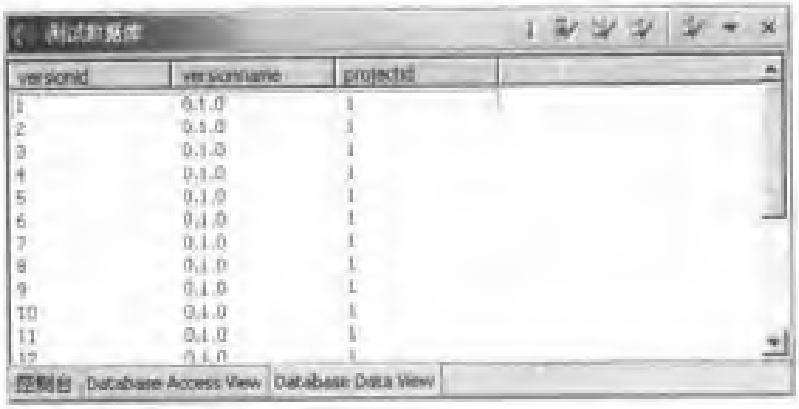


图 10-7　显示表数据的视图

编辑器主要是对一个 Hibernate Mapping File 进行可视化的编辑，如图 10-8 所示。



图 10-8　Hibernate 可视化编辑

2. Tanghan 插件的主要功能

主要功能有如下三项：

（1）浏览表结构，表数据；
（2）执行简单 SQL；
（3）数据可以设置分页显示；
后续功能正在开发过程中。

10.3.2　Tanghan 插件下载及安装

Tanghan 插件下载路径为 http://forum.hibernate.org.cn/download/jlinux/，现在支持 Eclipse 3.0 的插件也发行了，发行包名是 com.tanghan.plugin_0.1.0.3m5.12.21.rar 本次将下载文件是 com.tanghan.plugin_0.1.0.12.21.rar 发行包支持 Eclipse2.1 版本。Tanghan Plugin 的安装很简单，解压后将文件包放到 Eclipse 安装路径下的 plugins 文件夹下面，重新启动 Eclipse。打开“窗口（W）”→“显示视图（V）”→“其它（O）”，在图 10-9 显示的画面中，选择“Tanghan Plugin”下面的两项“Database Access View”和“Database Data View”。按“确定”后，在 Eclipse 控制台的位置上显示图 10-10 的画面。该图是显示数据库连接和驱动的视图，主要用于记录有哪些数据库驱动和与数据库的连接状态。其中 表示该驱动已经被加载， 表示该驱动无法被加载。



图 10-9　显示视图



图 10-10　数据库连接视图

10.3.3　使用 Tanghan 插件连接数据库

1．加载 JDBC 驱动

在加载 JDBC 驱动之前，必须有与该数据库相适应的 JDBC 驱动程序。可以从下面的 URL 中得到 MySQL 的 JDBC 驱动程序，http://www.mysql.com/downloads/api-jdbc-stable.html。本次介绍使用 3.0.9 版本。从上述地址下载 mysql-connector-java-3.0.9-stable.zip，然后将解压后的 mysql-connector-java-3.0.9-stable-bin.jar 文件拷贝到 JBoss 安装路径\server\default\lib 下，本机指定的路径为 H:\jboss-3.2.2\server\default\lib 下。

可以选择新加入一个 JDBC 驱动，也可以选择修改一个 JDBC 驱动。在这里选择修改 MySQL 的 JDBC 驱动，具体方法可参考图 10-10，“驱动”→在 MySQL 上右击鼠标，选择“修改驱动程序信息”，在弹出的对话框中，输入 JDBC 驱动的相关信息：

驱动名称　　MySQL（可以是自己命名）
驱动程序包名　　H:\jboss-3.2.2\server\default\lib\
　　mysql-connector-java-3.0.9-stable-bin.jar
驱动程序类　　com.mysql.jdbc.Driver
驱动程序前缀　　jdbc:mysql://localhost/mldb（//<host>：<post>/<DBName>）

端口号也可以省略不写。输入完信息的情况如图 10-11 所示。



图 10-11　“添加新的驱动”对话框

按“确定”按钮后，该驱动便被加载上去了。

2. 添加数据库连接

添加一个数据库连接可以按以下步骤完成，参考图 10-10，在“数据库”上右击鼠标，选择“添加新的数据库”，弹出的对话框如图 10-12 所示。在该对话框中输入 JDBC 驱动的如下信息：

数据库连接名　　MySqlTest（自己命名）
驱动名称　　MySQL（选择下拉框中已经加载上的驱动名称）
驱动程序类　　com.mysql.jdbc.Driver
数据库连接　　jdbc:mysql://localhost/mldb
用户名　　mailAdmin（指在第 5.1 节里建的数据库 mldb 的全权用户）
密码　　mailAdmin

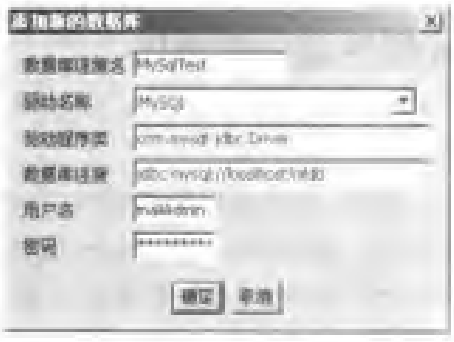


图 10-12　“添加新的数据库”对话框

3. 数据库连接

选中新建的数据库连接“MySqlTest”右击鼠标，选择“连接数据库”，如图 10-13 所示。如果一切设置正确的话，会在“MySqlTest”左边出现一个“+”号，表示该数据库已经正常连接上了。

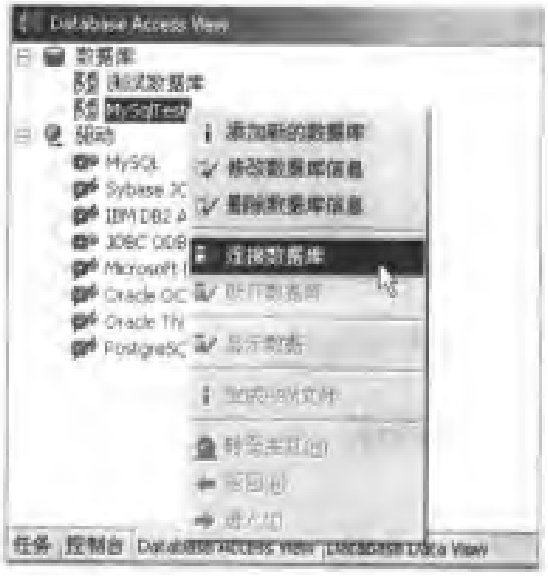


图 10-13　连接数据库

10.3.4　使用 Tanghan 插件进行数据库查询及修改操作

数据库连接上以后，双击该数据库，选中一个表，右击鼠标，在弹出的上下文菜单中选择“显示数据”，如图 10-14 所示。这时数据会在另外一个视图中出现。在该视图中，不仅能够看到表的数据，还可以进行简单的 SQL 语句的编写。SQL 语句编写完成后，点鼠标右键执行 SQL，如图 10-14 所示。接下来进行数据的插入。以下是 insert 语句：

```
insert into mlember values("Tom@sohu.com ", "Tom");
```

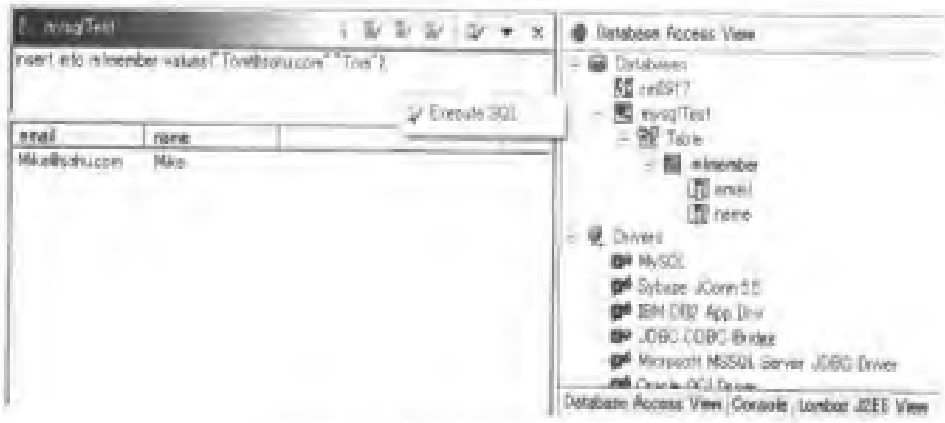


图 10-14　查看表及执行 SQL 语句

若要修改浏览数据库数据时使用的分页信息，则
选择“窗口”→“首选项”→“Tanghan Plugin”，打开如图 10-15 所示页面。



图 10-15　修改浏览数据库数据时使用的分页信息

10.3.5　数据库连接测试程序

1．创建一个类（Class）

在 J2EE_test/myWeb 上右击鼠标新建一个名为 dbConn 的类，将上节中 dbConn.java 类的源代码拷贝到该类中。

2．将 mysql 驱动程序导入 J2EE_test 项目中

在 J2EE_test 项目上右击鼠标，选择“属性”→“Java 构建路径”→“添加外部 JAR（X）”，选择 H:\jboss-3.2.2\server\default\lib\路径下的 mysql-connector-java-3.0.9-stable-bin.jar，如图 10-16 所示。



图 10-16　将 MySQL 驱动程序导入“J2EE_test”项目中

3．运行 dbConn.java 类

选择“运行（R）”→“运行为（S）”→“Java 应用程序”，如果数据库连接正确，将看到如图 10-17 所示的运行结果。

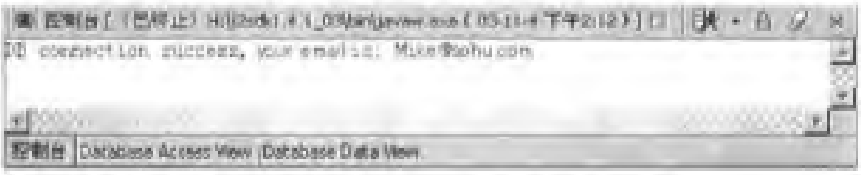


图 10-17　运行 dbConn.java 类

10.4　EJB 与数据库的连接

上一节介绍了一个简单的 Java 应用程序与数据库连接的例子，下面将要介绍 EJB 与数据库连接的方法，包括 JBoss 服务器上数据库的设定、实体 Bean 做成以及 EJB 与数据库连接的测试等。

10.4.1　JBoss 的数据库设定

通常需要将 JBoss 上运行的 EJB 连接到数据库上，下面将介绍如何在 JBoss 上进行 JDBC 驱动以及数据库的设定。

1．JDBC 驱动程序的安装

在 JBoss 和数据库连接时，必须有与该数据库相适应的 JDBC 驱动程序。在 10.3.3 节中已经将驱动程序 mysql-connector-java-3.0.9-stable-bin.jar 文件拷贝到 JBoss 安装路径 \server\default\lib 下，本机指定的路径为 H:\jboss-3.2.2\server\default\lib。

2．数据源（DataSource）的设定

（1）拷贝 MySQL 使用的设定文件，并对其进行编辑，设定数据源。

将 H:\jboss-3.2.2\docs\examples\jca\mysql-service.xml 文件拷贝到 H:\jboss-3.2.2\server\default\deploy 文件夹下面。

（2）修改拷贝到 deploy 文件夹下面的 mysql-service.xml 文件。

在大概 50 行的位置作以下修改：指定连接目标数据库。这次连接到本机的数据库 mldb 上需要进行如下设定：

```
    <config-property name="ConnectionURL" type="java.lang.String">
jdbc:mysql://localhost/mldb
    </config-property>
```

在大概 51 行的位置作以下修改：原来默认指定的驱动程序是 org.git.mm.mysqlDriver，这是旧版的驱动程序。现在需要做以下的修改：

```
    <config-property name="DriverClass" type="java.lang.String">com.mysql.jdbc.
```

```
Driver</config- property>
```

在大概 53~54 行的位置作以下修改：指定用户名和密码。现在指定先前做成的 mailAdmin 用户。

```
    <config-property name="UserName" type="java.lang.String">mailAdmin</config-
property>
    <config-property name="Password" type="java.lang.String">mailAdmin</config
-property>
```

10.4.2 生成实体 Bean

本节通过创建一个实体 Bean，说明 EJB 如何能够与数据库相连接。实体 Bean 是能够存放在永久性存储空间的持久对象。实体 Bean 包括 BMP 和 CMP，其中 BMP 有其自身的 SQL 语句，因此使用起来可能会有些麻烦。下面只介绍使用起来简单方便的 CMP。

现使用 Lomboz 来做成实体 Bean。本节接续前一章所作成的 J2EE_test 的项目，可以省略新建 J2EE 项目这一步，直接从设定数据源的操作开始进行。

1. 设定数据源

首先修改 J2EE_test\myEJB\META-INF\路经下的文件 ejbGenerate.xml，然后进行 JBoss 的数据源的设定。在原文件对应“PLEASE_MODIFY_THIS”的部分，做如下修改（斜体部分为修改部分）：

```
<jboss
   version="3.0"
   unauthenticatedPrincipal="nobody"
   xmlencoding="UTF-8"
   destdir="${ejb.dd.dir}"
   validatexml="false"
   datasource="java:/MySqlDS"
   datasourcemapping="mySQL"
   preferredrelationmapping="foreign-key"
/>
```

2. 生成 EJB

使用 Wizard 生成实体 Bean 类。在“J2EE_test”上右击鼠标，选择“新建（W）”→“其它（O）”→“Lomboz EJB Creation Wizard”，打开对话框，然后输入源文件夹名、包名及类名等信息，最后在“Select EJB Type”中选择“Container Managed Entity EJB”类型即可，如图 10-18 所示。

此时单击“下一步（N）”，会出现设定 CMP 时所使用的对话框。现做如下设定。

在 Persistent fields 框里添加如下内容：

email	
Field	email
Field type	java.lang.String
Database column	email
SQL type	VARCHAR

（续表）

Name	
Field	name
Field type	java.lang.String
Database column	name
SQL type	VARCHAR

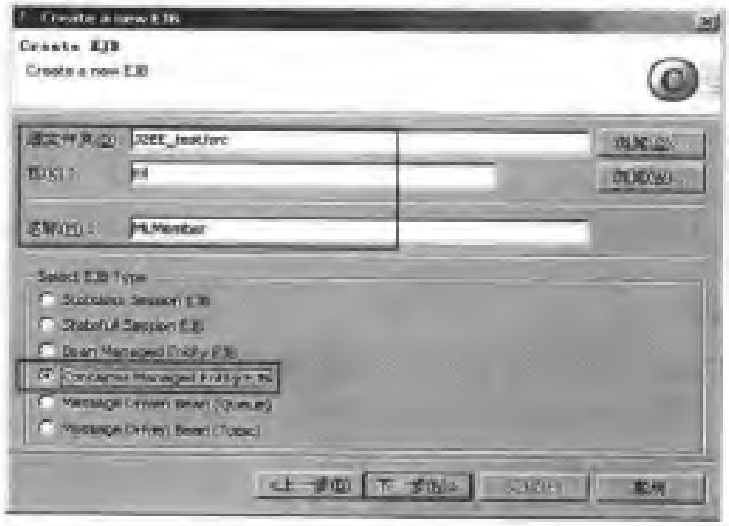


图 10-18　生成实体 Bean 类

填写完 email 的信息后，按“Add”按钮，则在 Fields 列表中添加了 email 的相关信息，接着再填写 name 的信息。从 Fields 列表中选中 email，然后单击“Make primary key”按钮，这样就把 email 设置为主键了。设置后的情况如图 10-19 所示。

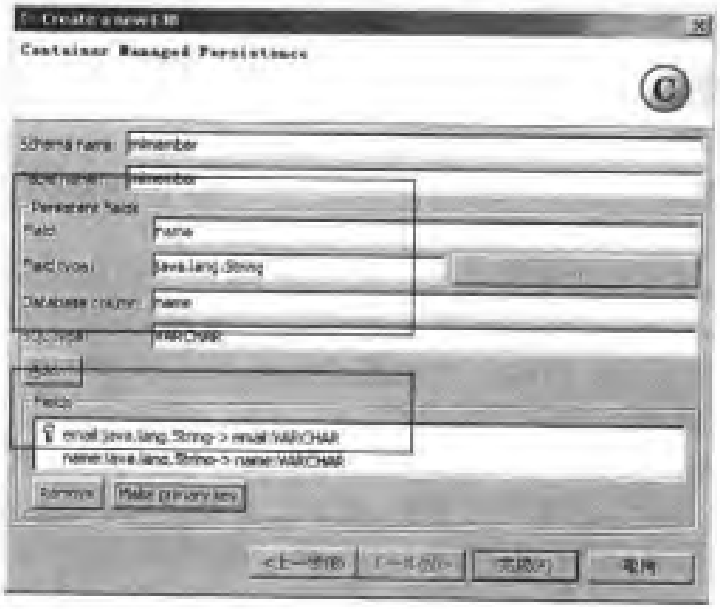


图 10-19　设定 CMP 的对话框

单击“完成（F）”按钮后，会在“J2EE_test/src/ml”文件夹下面生成一个新的 Java 文件。本例生成的类名为 MLMemberBean.java。

打开所生成的 MLMemberBean.java 文件，其注释部分如下所示：

```
/**
* @ejb.bean name="MLMember"
jndi-name="MLMemberBean"
* type="CMP"
* primkey-field="email"
* schema="mlmember"
* cmp-version="2.x"
*
* @ejb.persistence
* table-name="mlmember"
*
* @ejb.finder
* query="SELECT OBJECT(a) FROM mlmember as a"
* signature="java.util.Collection findAll()"
**/
```

3. 编辑源代码

首先修改 ejbCreate 方法，该方法用来记述生成实体 Bean 时的初期编码。修改 TODO: 所指定的部分。下面对 ejbCreate 方法作以下修改：

- 在参数中追加 email, name。
- 在方法内部，调用 setEmail, setName 的方法。

ejbCreate 方法修改后的源代码如下：

```
public java.lang.String ejbCreate(String email, String name) throws
  javax.ejb.CreateException {
      // EJB 2.0 spec says return null for CMP ejbCreate methods.
      // TODO: YOU MUST INITIALIZE THE FIELDS FOR THE BEAN HERE.
      // setMyField("Something");
      setEmail(email);
      setName(name);
      return null;
}
```

4. 生成 EJB 类

同第 9 章所讲到的生成无状态会话 Bean 一样，实体 Bean 也可以通过使用 Lomboz 中的 XDoclet 自动生成一个实体 EJB 所需要的 Java 文件。

（1）将 Java 文件追加到 EJB 模块中。在“MLMemberBean.java”上右击鼠标→“Lomboz J2EE...”→“Add EJB to module...”。在所显示出来的对话框中，单击“Ejb[myEJB]”，然后单击“OK”。这时打开 beans.xml 和 ejbs.xml 文件，就能看到 MLMemberBean 已经被追加到这些文件中了。

（2）生成 EJB 的文件。在“myEJB”上右击鼠标→“Lomboz J2EE...”→“Generate EJB Classes”。在 ejbsrc 文件夹的下面，生成了如图 10-20 所示的文件。另外，在 myEJB/META-INF 文件夹下，自动更新或者生成了适应各类服务器的部署描述符，并且同时生成了 Stateless Session Bean 时没有生成的、与 CMP 相关联的描述符。



图 10-20　生成 EJB 的文件目录

10.4.3　实体 Bean 的部署和运行

下面对在第 9 章完成的无状态会话 Bean（MLAdminBean.java）进行一些修改，使其能够操作实体 Bean，从数据库中取出名字。为此，需要进行 XDoclet 标记的修正，类源代码的变更以及 EJB 的再生成这几个步骤。

1．XDoclet 标记的修正

为了达到从 MLAdmin 可以直接调用 MLMember 的目的，现将“@ejb.ejb-ref”作为 XDoclet 标签追加到 MLAdminBean.java 的类中（斜体部分为追加部分）。

```
/**
* @ejb.bean name="MLAdmin"
* jndi-name="MLAdminBean"
* type="Stateless"

* @ejb.ejb-ref
```

```
 * ejb-name="MLMember"
 **/
```

根据这个标签，当生成 EJB 类时，在与 ejb-jar.xml 的 MLAdmin 相关的记述中，记述了与 MLMember 有关的信息。于是就可以利用 MLAdmin 中被称为“java:comp/env/ejb/MLMember”的 COMP_NAME，来参考 MLMember 了。以下这段代码是修改标记前的 ejb-jar.xml 中与 MLAdmin 相关的代码，读者可以用它来对比标记修改后重新生成 EJB 以后的 ejb-jar.xml 中与 MLAdmin 相关的代码段。

修改前的 ejb-jar.xml 部分代码：

```
<session >
   <description><![CDATA[]]></description>
   <ejb-name>MLAdmin</ejb-name>
   <home>ml.MLAdminHome</home>
   <remote>ml.MLAdmin</remote>
   <local-home>ml.MLAdminLocalHome</local-home>
   <local>ml.MLAdminLocal</local>
   <ejb-class>ml.MLAdminSession</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>

</session>
```

2. 修改 MLAdminBean.java 类

将 MLAdminBean.java 类中 getName 方法的内容，按如下所示进行变更。该方法是当邮件地址在数据库中不存在时，就会返回一串错误提示“NO SUCH AN EMAIL ADDRESS!”。其他情况下，则抛出异常。

以下是 MLAdminBean.java 类中 getName 方法的内容：

```
/**
* @ejb.interface-method
*   tview-type="remote"
**/
public String getName(String email){
    try {
        MLMemberHome home = MLMemberUtil.getHome();
        return home.findByPrimaryKey(email).getName();
    } catch (FinderException e) {
        e.printStackTrace();
        return "NO SUCH AN EMAIL ADDRESS!";
    } catch (NamingException e) {
        e.printStackTrace();
        throw new EJBException(e);
    } catch (RemoteException e) {
        e.printStackTrace();
        throw new EJBException(e);
    }
}
```

3. EJB 的再生成

因为刚刚对 XDoclet 标签作了修改，所以需要重新生成 EJB。右击“myEJB”文件夹→“Lomboz J2EE...”→“Generate EJB Classes”。另外还要确认一下 ejb-jar.xml 文件

中有关 MLAdmin 的记述中，是否追加了 MLMember 所使用的 ejb-ref 要素。

以下是 EJB 再生成后的 ejb-jar.xml 部分代码（斜体部分是修改标记后追加上去的代码）：

```
<session >
  <description><!"CDATA""></description>
    <ejb-name>MLAdmin</ejb-name>

    <home>ml.MLAdminHome</home>
    <remote>ml.MLAdmin</remote>
    <local-home>ml.MLAdminLocalHome</local-home>
    <local>ml.MLAdminLocal</local>
    <ejb-class>ml.MLAdminSession</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>

    <ejb-ref >
      <ejb-ref-name>ejb/MLMember</ejb-ref-name>
      <ejb-ref-type>Entity</ejb-ref-type>
      <home>ml.MLMemberHome</home>
      <remote>ml.MLMember</remote>
      <ejb-link>MLMember</ejb-link>
    </ejb-ref>


</session>
```

4. 部署和程序运行

将追加的实体 Bean 以及修改后的无状态会话 Bean 部署到 JBoss 中，并运行程序进行测试。

（1）部署 EJB。在“myEJB”文件夹上右击鼠标→“Lomboz J2EE...”→“Deploy Module”即可。在 myEJB.jar 文件中，包含 MLAdmin 和 MLMember 两部分的 EJB。由于 JBoss 对应的是热部署，所以在 JBoss 启动进行部署时，在 JBoss 的控制台上会显示如下信息：

```
12:07:16,140 INFO  [MainDeployer] Starting deployment of package: file:/H:/
   jboss-3.0.8_tomcat-4.1.24/server/default/deploy/myEJB.jar
12:07:16,296 INFO  [EjbModule] Creating
12:07:16,328 INFO  [EjbModule] Deploying MLMember
12:07:16,359 INFO  [EjbModule] Deploying MLAdmin
12:07:16,437 INFO  [EjbModule] Created
12:07:16,437 INFO  [EjbModule] Starting
12:07:16,640 INFO  [MLMember] Table 'mlmember' already exists
12:07:16,656 INFO  [EjbModule] Started
12:07:16,671 INFO  [MainDeployer] Deployed package: file:/H:/jboss-3.0.8_
   tomcat-4.1.24/server/default/deploy/myEJB.jar
```

这样用户就可清楚地知道 MLMember 和 MLAdmin 两者的 EJB 都进行了编译。

（2）运行 EJB。当 JBoss 没有启动时，在“myEJB”文件夹上单击鼠标右键→“Lomboz J2EE...”→“Debug Server”，可以实现 JBoss 的启动。接下来可以试着由客户端进行连接。选择第 9 章中做成的 MLTest.java 文件，然后选择“Java 应用程序”方式运行。这样，在控制台上就可以看到显示出的执行结果，见图 10-21。



图 10-21　运行 EJB 的执行结果

本章小结

本章主要讲解了 MySQL 数据库的构建及一些基本的指令，如新建数据库和表及插入数据等，还讲解了 Java 数据库连接——JDBC，包括 JDBC 驱动类型与几种常用数据库建立连接的方法等，并举了一个连接数据库的 Java 应用程序的例子。另外，还介绍了在 Eclipse 上使用 Tanghan Plugin 来建立数据库连接，最后通过一个实体 Bean 的例子，着重讲解了 EJB 与数据库的连接方法。

第七篇

Eclipse 高级应用

PART 7

- 第 11 章　Eclipse 环境下的 Struts 框架应用
- 第 12 章　将 Eclipse 作为建模工具
- 第 13 章　使用 Eclipse 进行重构
- 第 14 章　Eclipse 插件开发
- 第 15 章　Eclipse 的界面 API

第11章　Eclipse环境下的Struts框架应用

本章将向读者介绍使用 Eclipse 和一个开源的基于 JSP 的框架 Struts 来开发 Web 应用程序。通过两个详细的示例，使读者了解 Struts 的结构和其工作的过程。

具体内容包括：

- Struts 框架的结构与组成
- MVC 模式结构
- Struts 标签库
- Easy Struts 插件介绍
- 使用 Struts 框架开发应用程序

11.1　Struts 框架简介

Struts 是一个在 Java Servlet 和 JavaServer Page 技术中被用来构建 Java Web 应用程序的开源框架。是 Jakarta Project 的一部分。

Struts 基于 MVC（Model View Controller）设计模式，支持基于 Model 2 的应用程序结构。Struts 本身提供了自己的控制（Controller）组件并且整合其他技术所提供的模型（Model）和视图（View）。对于模型（Model）来说，Struts 可以很好的与标准的数据访问技术互动，像 JDBC 和 EJB。对于视图（View），Struts 可以和 JavaServer Page 很好的工作。Struts 将一个应用程序的业务逻辑，控制逻辑和表现逻辑分解，使应用程序的重用性和维护性更好。

11.2　MVC 模式

11.2.1　MVC 模式结构

MVC，被用来构建用户界面，这种设计思想来源于 Smalltalk。使用 MVC 设计模式构建的应用程序被分为三层或者说被分成了三种对象——模型（Model）对象、视图（View）对象和控制（Controller）对象。MVC 设计模式比平常所说的 GOF 的 23 种设计模式的范围要广，在 MVC 中使用了 23 种模式中的部分模式(比如 Observer, Composite 和 Strategy 等)。

下面来详细介绍一下这三种对象。

- 模型（Model）是应用程序的核心。用来完成应用程序的主要功能，模型接收由视图发出的请求，并将处理请求的结果返回给视图，虽然请求是由视图发出的，但是这个请求是由控制对象来转发的，所以模型所进行的操作相对于视图来说是黑箱操作。模型可以用来封装应用程序的状态和数据，状态用来为控制对象对模型的调用和操作提供信息，数据是在视图中向用户展示的内容。
- 视图（View）用来显示模型的内容。是显示在用户面前用来与用户进行交互的应用程序的外观。可以理解为 HTML 页面，当然也可以是 JSP，XML，Macromedia Flash，Web Services 等。视图应当只是用来收集数据和显示模型，以及将用户的请求转交给控制对象，而不应当参与模型中业务流程的处理。同一个业务流程可能会由于处理结果的不同而对应不同的视图。
- 控制（Controller）对用户的请求作出反应，创建和设置模型，并将模型与视图进行协调，来共同完成用户的请求。可以将控制对象理解为一个分发器，它会根据不同的用户请求来选择不同模型和视图。同样，控制对象也不会对模型中的业务流程产生影响。

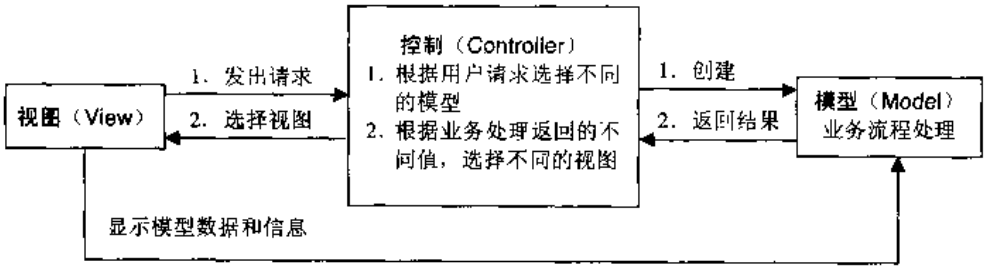


图 11-1　MVC 模式

11.2.2　Model 2 体系结构

Model 2 表示的是基于 MVC 模式的框架。Model 2 体系结构如图 11-2 所示，是一种把 JSP 技术与 Servlet 联合使用来实现 Web 服务的方法。它使用 Servlet 来充当控制（Controller）对象的角色，负责对用户请求的接收和处理，通过对用户请求的判断来决定创建哪一个 JavaBean 对象，并且决定回应用户请求的应该是哪一个 JSP。Servlet 创建好 JavaBean 后，JavaBean 通过调用 EJB 和数据库等对象组件来收集数据，并将结果通知 Servlet，由 Servlet 决定将要转到哪一个 JSP。JSP 通过 JavaBean 中的数据来生成显示给用户的内容，在 JSP 页内没有业务逻辑，它仅负责检索由 Servlet 创建的对象或 JavaBean，从中提取用于显示的内容。正是因为 Model 2 将一个应用程序的业务逻辑，控制逻辑和表现逻辑分解，所以明确了开发人员和网页设计人员的分工，提高了工作的效率，也使得应用程序的重用和维护变得更加容易。对于复杂的项目而言，使用 Model 2 体系结构的好处是显而易见的。

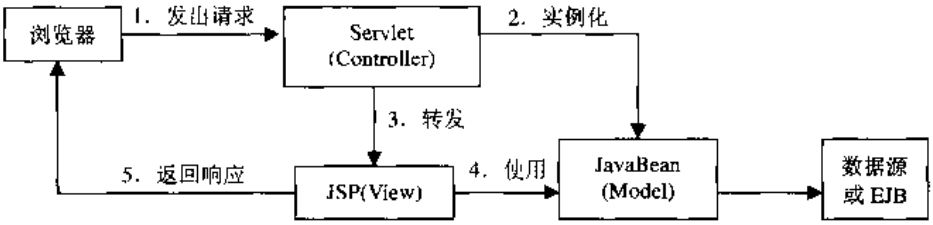


图 11-2　Model 2 体系结构

11.2.3　MVC 设计模式的小结

MVC 最突出的优点是使得多个不同视图可以共享同一个模型。由于将数据和逻辑从表示层中分离了出来，所以无论用户想要的是什么样的界面，如 HTML 或者是 Flash，都可使用同一个模型对不同的界面进行支持，这同时也实现了代码的重用。由于模型返回的数据没有进行格式化，所以同样的构件能被不同界面使用。如果想把数据库从 MySQL 移植到 Oracle，只需改变模型即可。只要正确的实现了模型，不管数据来自数据库或是 LDAP 服务器，视图都会正确的显示它们。由于运用 MVC 的应用程序的三个部件相互独立，因此改变其中一个不会影响其他两个，所以依据这种设计思想能构造良好的低藕合的构件。

有相当一部分的基于 Web 的应用程序都是使用像 ASP 或者 PHP 这样的过程化语言来创建的。它们会将表示层代码和数据库查询之类的数据层代码混合在一起。有的开发者会试图将数据与表示层剥离开来，但通常不是很容易做到的，它需要精心的计划和大量的测试。MVC 将它们强制性的分开，这样会带来一些额外的工作，但是对于 MVC 所带来的好处而言，这些工作都是值得的。

11.3　Struts 的下载和安装

下载参见 Struts 的官方网址：http://jakarta.apache.org/struts。

Struts 的二进制码和源代码的下载地址如下：

二进制码为 http://jakarta.apache.org/site/binindex.cgi。

源代码为 http://jakarta.apache.org/site/sourceindex.cgi。

读者可以根据自己的开发平台下载相应的压缩文件。如果下载的是二进制的压缩文件，解压后的文件夹结构如图 11-3。



图 11-3　文件夹结构

下面介绍各文件夹的内容，这里面包括：

- contrib 文件夹中是一些扩展的 Struts 的自定义标签库。
- lib 文件夹中包括 struts.jar 等库文件，和 Struts 自定义标签库描述文件（*.tld）。
- webapps 文件夹中包含了许多使用 Struts 框架的 Web 的应用程序的示例。其中包括以下八项内容。
 - struts-blank.war：一个空的 Web 应用程序，用户可以使用它作为基础来开发自己的应用程序。
 - struts-upload.war：如何使用 Struts 框架上传文件的示例。
 - struts-documentation.war：Web 应用文档，包括了所有的 Struts 的文档。
 - struts-example.war：一个使用了大量的 Struts 特性的 Web 应用的示例。
 - struts-exercisetaglib.war：这个应用程序包含了 Struts 所支持的用于检测用户自定义标签的网页，也可以用来示例如何使用 Struts 标签。
 - struts-template.war：Struts 模板标记的介绍和使用示例。
 - struts-validator.war：如何在服务器端和客户端进行数据校验的示例。
 - tiles-documentation.war：如何使用 tiles 的 Web 应用程序的示例。

如果下载的是 Struts 的源代码，需要在 Struts 的解压目录中使用 build.bat 脚本文件编译源代码。通常使用 Ant 来编译源代码（可以参考第 6 章）。

11.4 一个简单的 Struts 程序

现在利用 11.3 节中提到的空的 Web 应用程序（struts-blank.war），来尝试着做一个简单的 Web 应用程序。通过这个程序先简单的了解一下 Struts 的工作过程。

1. 新建 Tomcat 工程

首先，用 Eclipse 做一个 Tomcat 项目（具体做法请参考第 8 章）。在这里暂时命名为 struts_Test01。

注意：在创建工程的时候，请不要忘记选择“更新 server.xml”的选项。

2. 保存 struts-blank.war

将上面提到的 struts-blank.war 解压后的文件拷贝到 Eclipse 安装目录下的 workspace /struts_Test01/这个文件夹中，见图 11-4。



图 11-4 文件夹结构

3. 启动 Tomcat

启动机器上的 Tomcat，打开浏览器，键入 URL：http://localhost:8080/struts_Test01/index.jsp 确认一下 index.jsp 是否启动。如果看到“Welcome!”等字样，就说明 index.jsp 已经启动。如果看到了 index.title 等资源未找到的错误提示，请试着修改 index.jsp，比如把它修改成：

```
<html>
<head></head>
<body>Hello </body>
</html>
```

再试一次上面的网址，就可以看到写着 Hello 的网页了。

4. 要做什么

在编写程序代码之前，首先应当清楚要编写什么，本例中要创建或者编辑：

- HelloAction.java：Action 类。
- hello.jsp：用来显示的 JSP 文件。
- struts-config.xml：管理画面之间相互迁移的文件。

5. 设定 Struts 的 ClassPath

在 Eclipse 中，以鼠标右键单击刚创建的工程 struts_Test01→“属性（R）”（如图 11-5 所示）。



图 11-5　选择属性选项

在属性画面中选择“Java 构建路径”，然后选择“库（L）”选项卡，按下“添加 JAR...”（如图 11-6 所示）。



图 11-6　添加 JAR 文件

将/WEB-INF/lib 下的所有的*.jar 文件都添加进来（如图 11-7 所示）。



图 11-7　选择所有文件

设定 Class Path 是编译的要求，使得 Eclipse 在编译程序的时候可以找到 Struts.jar。

6. 编写 Action 类

现在使用 Eclipse 创建类文件，需要创建的类只有一个“test.HelloAction”。

首先，新建这个类（请参考第 3 章），在新建的项目 struts_Test01 上单击右键→“新建（W）”→“其它（O）”。在新的窗体中选择“类”→“下一步”，在“包（K）”这一项中填写“test”，“名称（M）”中填写“HelloAction”，单击“完成（F）”。建好后，Eclipse 如图 11-8 所示。



图 11-8　Eclipse 工作台

这个类需要继承 org.apache.struts.Action 这个类，所以要添加上 extends Action，写成如图 11-9 所示的形式。

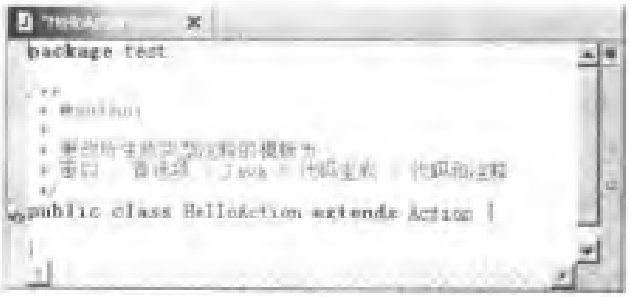


图 11-9　继承 Action 类

Eclipse 在 Action 下面的红线和这一行前面的错误提示，是因为还没有导入 Action 这个类。Eclipse 提供了一个简单的不需要手写的导入父类的方法：在编辑窗口上右击鼠标，选择“源（S）”→“组织导入（O）”（如图 11-10）



图 11-10　组织导入

选择“org.apache.struts.action.Action”即可，将会看到导入的类，如图 11-11 所示。

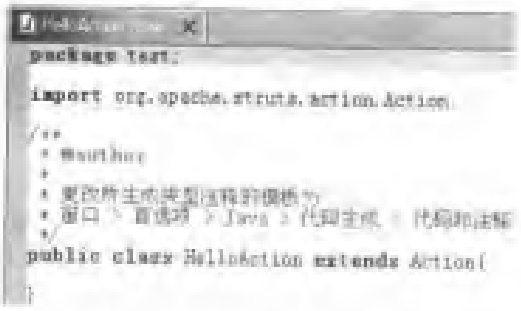


图 11-11　导入 org.apache.struts.action.Action

接下来，要向 HelloAction 类中添加方法。在编辑窗口中右击鼠标，选择“源（S）”→“覆盖/实现方法（V）”（如图 11-12 所示）。

选中“execute(ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse)”方法，单击“确定”（如图 11-13 所示）。

这样，就把这个方法追加到程序中了（如图 11-14 所示）。

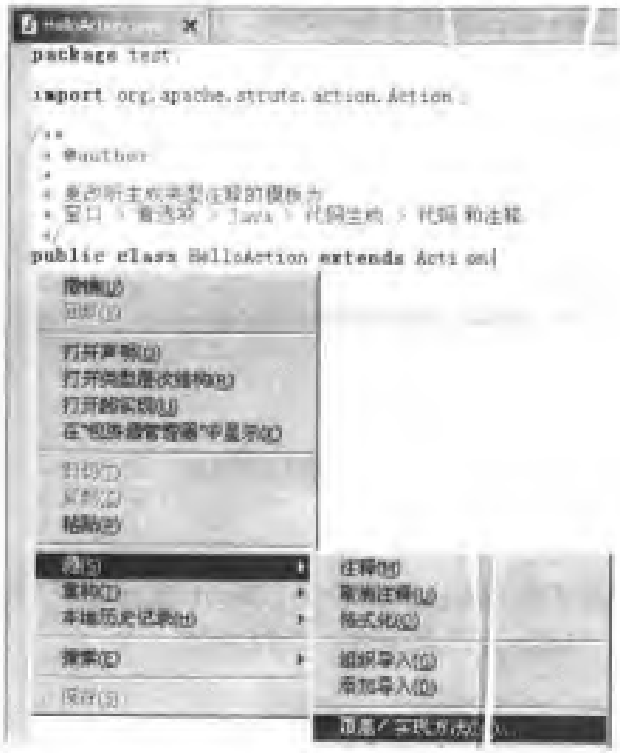


图 11-12　覆盖/实现方法

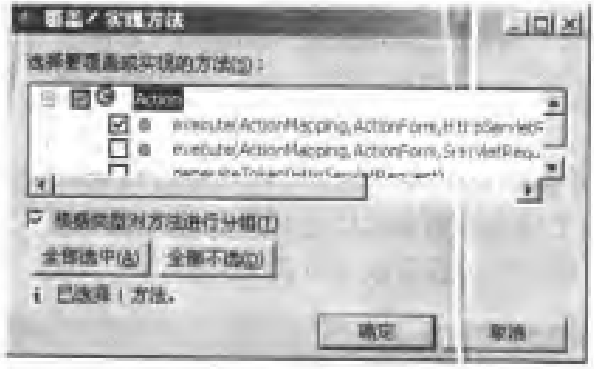


图 11-13　选择 execute()方法

```
public class HelloAction extends Action{

    public ActionForward execute(
        ActionMapping arg0,
        ActionForm arg1,
        HttpServletRequest arg2,
        HttpServletResponse arg3)
        throws Exception {
        // TODO 自动生成方法存根
        return super.execute(arg0, arg1, arg2, arg3);
    }

}
```

图 11-14　自动生成代码

下面，稍微修改一下这个类。添加一条后台打印语句。再修改一下 return 语句，返回“success”。Struts 就是通过这个被返回的“success”来判断应该向哪一个 JSP 页面迁

移。不要忘了保存，Eclipse 会在保存文件的时候，自动地编译文件。修改后的代码如图 11-15 所示。

```
public class HelloAction extends Action{

    public ActionForward execute(
        ActionMapping arg0,
        ActionForm arg1,
        HttpServletRequest arg2,
        HttpServletResponse arg3)
        throws Exception {
        System.out.println("Hello Struts!");
        return arg0.findForward("success");
//      return super.execute(arg0, arg1, arg2, arg3);
    }

}
```

图 11-15　修改后的代码

7. 创建迁移目标的 JSP

在 Eclipse 安装目录下的 workspace/struts_Test01/pages/文件夹（如果没有就创建一个）中，创建一个名叫“hello.jsp”的文件。其内容如下：

```
<html>
  <head></head>
  <body>Hello Struts! </body>
</html>
```

8. 编辑 struts-config.xml 文件

现在需要修改一下位于 workspace/struts_Test01/WEB-INF/下的 struts-config.xml 文件。修改这个文件是为了让 Struts 根据 HelloAction 类的返回值，来迁移到希望的 JSP 页面。

可以使用任何的文本编辑器来修改这个文件，当然 Eclipse 也可以修改，方法如下。

在 struts-config.xml 文件上右击鼠标，选择“打开方式（H）”→“文本编辑器”，就可将这个文件在 Eclipse 中打开进行修改（见图 11-16）。

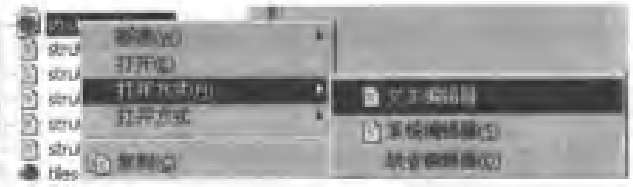


图 11-16　使用文本编辑器打开文件

找到“action-mappings”这个标签（大概在 205 行附近），在它的下面添加如下内容：

```
<action path="/hello" type="test.HelloAction">
    <forward name="success" path="/pages/hello.jsp" />
</action>
```

文件中，被修改部分的内容大致如图 11-17 所示。

```
<action-mappings>
        <!-- Default "Welcome" action -->
        <!-- Forwards to Welcome.jsp -->
    <action path="/Welcome"
            type="org.apache.struts.actions.ForwardAction"
            parameter="/pages/Welcome.jsp"/>

    <action path="/hello" type="test.HelloAction">
        <forward name="success" path="/pages/hello.jsp" />
    </action>
```

图 11-17　被修改的文件部分

这里，action 标签中的内容：path="/hello" type="test.HelloAction"，是告诉 Struts 如果遇到“hello”这个请求的话，请实例化 test.HelloAction，并运行其中的 execute 方法。forward 标签中的内容：name="success" path="/pages/hello.jsp"，是告诉 Struts 如果 test.HelloAction 这个类返回的值是“success”，那么请迁移到/pages/hello.jsp 这个 JSP 页中。当然，Action 类可以根据情况，返回多个不同的值，如果是这样，那么 forward 标签必须要将所有可能的情况都表示出来。例如，如果 test.HelloAction 这个类有可能返回两个值“success”和“false”，那么，struts-config.xml 中的内容应该做如下修改：

```
<action path="/hello" type="test.HelloAction">
  <forward name="success" path="/pages/hello_success.jsp" />
  <forward name="false" path="/pages/hello_false.jsp" />
</action>
```

9. 运行

重新启动 Tomcat，打开浏览器，键入 URL：http://localhost:8080/struts_Test01/hello.do，可以看到浏览器上显示“Hello Struts！”。

10. Hello Struts 的小结

在这个例子中，首先做了一个类 HelloAction，在这个类中完成了一些操作，这些操作是后台打印的语句（当然还可以完成更加复杂的操作），并且返回了一个 String 对象“success”。在实际的应用程序中，Action 类是最主要的类，大多数的工作都是由 Action 类来完成的。然后，做了一个用来显示的 JSP 文件，这里的例子比较简单，只是显示了一句话。在实际的应用程序中，JSP 更多的是用来显示由 Action 类从数据库中取到的数据，或者对数据库操作的结果。最后，还修改了/WEB-INF/下面的 struts-config.xml，这个文件主要是用来告诉 Struts 当一个 Action 类执行完操作后，应该迁移到哪一个 JSP 文件。一个 Action 类可以根据不同的情况返回不同的值，从而达到迁移到不同 JSP 文件的目的。

现在是否对 Struts 是如何工作的，以及 struts-config.xml 有何作用有了一定的了解呢？下面用一个流程图来说明 Hello Struts 的工作过程，如图 11-18 所示。

hello.do（浏览器发出请求）→Struts（根据 struts-config.xml 文件中的 action 标签的 type 属性来实例化 Action 类）→HelloAction（返回结果“success”）→Struts（根据 struts-config.xml 文件中的 forward 标签的 path 属性迁移到对应的 JSP 文件）→hello.jsp。

实际上，这个过程也就是为了实现将 View（JSP）和 Model（Action）分离的目的。由 Controller（Struts）来管理他们，以便于维护。

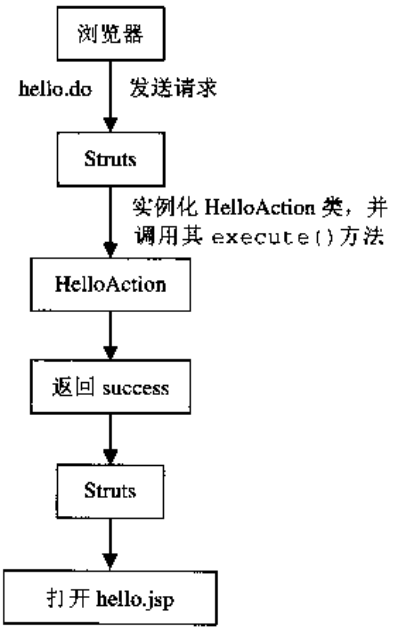


图 11-18　Hello Struts 的简单工作过程

如果要做的是一个很简单的应用（比如这个例子），只有少数几个页面，那么，完全可以不必使用 Struts 框架。但是，如果要做的是比较复杂的应用，有很多页面并且需要长期运行维护，那么，Struts 框架将是一个很好的选择。

11.5　Struts 的组成与结构

11.5.1　Struts 应用程序组件

下面来了解常用的 Struts 应用程序的组件，更多的内容请参考 Struts 的文档和 Struts 的源程序。在应用程序中最常被使用的组件包括以下几项。

1．ActionServlet 类

根据配置文件对用户请求进行调度的类。它的配置文件是 struts-config.xml。它继承 java.servlet.http.HttpServlet 类，JSP 文件中的请求是通过 ActionServlet 类的 doGet()和 doPost()方法得到的。这两个方法获得用户请求后，将请求传给 ActionServlet 类的 process()方法。在 process()方法中再将用户请求传递到 RequestProcessor 类的 process()方法中进行处理。在 RequestProcessor.process()方法中，主要完成了以下处理：

- 从请求中提取 URI 中不包括前缀和后缀的部分，也就是说，请求/login.do 被提取到的部分是/login。
- 把上一步中取得的 URI 作为关键字映射到 ActionMappings 中，如果在 ActionMappings 中有与之相符的 ActionMapping，那么就取得这个 ActionMapping 的实例。如果这个对象还没有被创建，那么就新建立一个这个对象的实例。

- 如果取得的 ActionMapping 的实例声明了 name 属性，那么 ActionServlet 会根据 name 属性来判断其声明的 ActionFrom 是否已经被创建。如果已经创建，则取得这个 ActionForm 的实例，否则创建一个新的 ActionForm 的实例。
- 通过 ActionMapping 的声明来判断取得的 ActionForm 类是否需要检验，如果需要，则调用 ActionForm 类的 validate()方法对 ActionForm 中的数据进行检验。如果检验结果没有问题，则继续进行下面的步骤。反之，如果没有通过检验，则返回发出请求的 JSP 文件。
- 找到用户请求的 Action 类，如果没有就新建一个 Action 类，并且把它保存起来。
- 调用 Action 对象的 execute()方法，将 request, response, ActionForm 和 ActionMapping 作为参数传递给 execute()方法，执行业务逻辑。业务逻辑执行完成之后，Action 类返回一个 ActionForward 对象。
- 接受由 Action 类返回的 ActionForward 对象，使用 RequestDispatcher.forward()方法或者 response.sendRedirect()方法，将 response 转发到 ActionForward 对象指定的资源，为用户显示 JSP 页面（当然也可以是转到另一个 Action 或者是 Servlet 中）。

2. ActionMapping 类

是在 struts-config.xml 中被定义的类，它反映的是用户请求中的 URI 和 ActionForm 以及 Action 类的实例之间的映射关系。它继承了 ActionConfig 类，在 ActionConfig 类中封装了 struts-config.xml 中定义的所有配置信息。需要注意的是，配置文件中的<action>元素中的 path 属性，也就是用户请求的 URI，必须以“/”开头。而且，任何扩展映射的扩展部分都将被删除。关于扩展映射的定义在 web.xml 文件中。下面的代码片段显示了扩展映射在 web.xml 中是如何被定义的。

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

上面这段代码的意思是，如果遇到以.do 结尾的用户请求，那么这个请求将被发送到 ActionServlet，ActionServlet 接受到这个请求后，会自动将.do 删除，并查找相应的 ActionMapping 来发送这个请求。例如，对于一个请求/login.do，ActionServlet 会将其转化为/login 后，寻找 path=/login 的 ActionMapping，并向它发送这个请求。

3. Action 类

是用来在具体的应用程序中描述商业流程的类。所有自己编写的 Action 类都必须继

承 org.apache.struts.action.Action 类，并且重载 Action 类中的 execute()方法。根据传递参数的不同，有以下两种重载的方法。

```
• public ActionForward execute(ActionMapping mapping,
                               ActionForm form,
                               ServletRequest request,
                               ServletResponse response)
      throws Exception

• public ActionForward execute(ActionMapping mapping,
                               ActionForm form,
                               HttpServletRequest request,
                               HttpServletResponse response)
      throws Exception
```

具体的应用程序中的业务逻辑流程是在 execute()方法中被描述的。ActionServlet 会为每个用户请求选择适当的 Action 类的实例，并调用其 execute()方法。这个 Action 类的实例类的目的就是来执行用户请求的业务逻辑。业务逻辑处理完成后，要给 ActionServlet 返回一个 ActionForward 对象。因为 ActionServlet 会对同时发生的请求共享同一个 Action 类的实例，所以对 Action 类的编程应该考虑线程安全的问题。Action 类中有一个 ActionServlet 类的实例变量，可以通过这个变量访问标准的 Servlet 资源。这个变量的值是这个 Action 类第一次被 ActionServlet 实例化的时候赋的值。Action 类中还保存了与用户请求相关的默认的 Locale 对象，可以通过方法：

```
protected Locale getLocale(HttpServletRequest request)
```

和

```
protected void setLocale(HttpServletRequest request, Locale locale)
```

对其进行读取或者设置。

4. ActionForm 类

是用来保存具体应用程序的数据和状态的类。它是由 ActionServlet 来实例化的。它还可以验证保存在其中的数据是否合法。一个 ActionForm 可以对应一个或多个 ActionMappings。在相应的 Action 类的 executes()方法被调用之前，ActionServlet 会自动完成下列任务。

- 在配置文件中的 scope 属性所定义的作用域内查找是否已经有了这个 ActionForm 的实例，如果没有这样的一个类的实例，则 ActionServlet 会自动建立一个新的 ActionForm 实例并将其加入到适当的作用域中。
- ActionForm 中的属性将会根据相应的用户请求中的内容被初始化。
- 如果配置文件中的 validate 属性被设置为 ture，那么在调用 executes()方法之前，ActionForm 中的 validate()方法将会被调用，用以校验 ActionForm 中数据的合法性。如果发现问题，那么 ActionForm 会将问题封装到一个错误对象中，并将错误返回给 ActionServlet，而 ActionServlet 将会根据配置文件中的信息，返回到 input 属性所指定的 JSP 文件。如果没有问题，那么 ActionFrom 将返回一个 null 对象，

这标志着所有的校验都通过了。

- ActionServlet 将经过校验（当然校验不是必需的）的 ActionForm 作为参数传递到 Action 类的方法 executes()中，这些值将会在业务逻辑的执行中被使用到。

5. ActionForward 类

是用来进行操作转移的类，可以通过在 Action 子类中返回 ActionForward 对象来转移到指定的 JSP。ActionForward 封装了配置文件中 forward 部分的信息。ActionForward 有两种类型：全局 ActionForward 和某个特定的 Action 的 ActionForward。在配置文件中对两种类型的 ActionForward 的定义分别如下：

```
<global-forwards>
     <forward  name="login"  path="/login.jsp"/>
</global-forwards>
<action-mappings>
   <action  path="/addStuff"
      type="test.example.addStuffAction"
      scope="request"
      validate="true"
      input="/add.jsp">
      <forward name="success" path="/finish.jsp"/>
   </action>
</action-mappings>
```

第一个 ActionForward 是全局的，而第二个则是属于一个具体的 Action 的。

6. ActionError 类

是 ActionForm 类返回的用于封装单独的错误信息的类。可以直接使用 String 作为 ActionError 类构造函数的参数，也可以在使用 String 作为参数的同时，指定若干的占位符共同作为 ActionError 类构造函数的参数。这两种方法的区别在于，保存在应用程序资源文件中的信息的不同。例如，资源文件中有一条错误信息：

```
error.message=<b>职员年龄应当大于 10 岁！ <b>
```

如果想在 JSP 文件中显示这句话，那么应该定义如下的 ActionError 类：

```
ActionError error = new ActionError("error.message ");
```

在 JSP 文件中，使用标签<html:error>就会在页面上显示出：

```
职员年龄应当大于 10 岁！
```

如果不想将上面信息中的 10 作为固定值，而希望在应用程序中根据具体情况来显示，那么，可以将资源文件中的错误信息修改为：

```
error.message=<b>职员年龄应当大于{0}岁！ <b>
```

这条信息中的“{0}”叫作占位符。在建立 ActionError 类的实例的时候，像下面这样建立：

```
ActionError error = new ActionError("error.message ", new Integer(20));
```

那么，在页面上显示的将会是：

```
职员年龄应当大于 20 岁!
```

当然还可指定更多的占位符，例如：

```
public ActionError(String key, Object value0, Object value1,Object value2,
                   Object value3)
```

7. ActionErrors 类

是 ActionError 类的容器。

8. Struts 标志库

是由 JSP 生成，用于在客户端浏览器显示的 HTML 文件。

11.5.2 MVC 模式下的 Struts

下面从 MVC 模式的角度了解 Struts 的相关的组件，下图虚线部分所示为 Struts 中和 MVC 模式相关的组件。

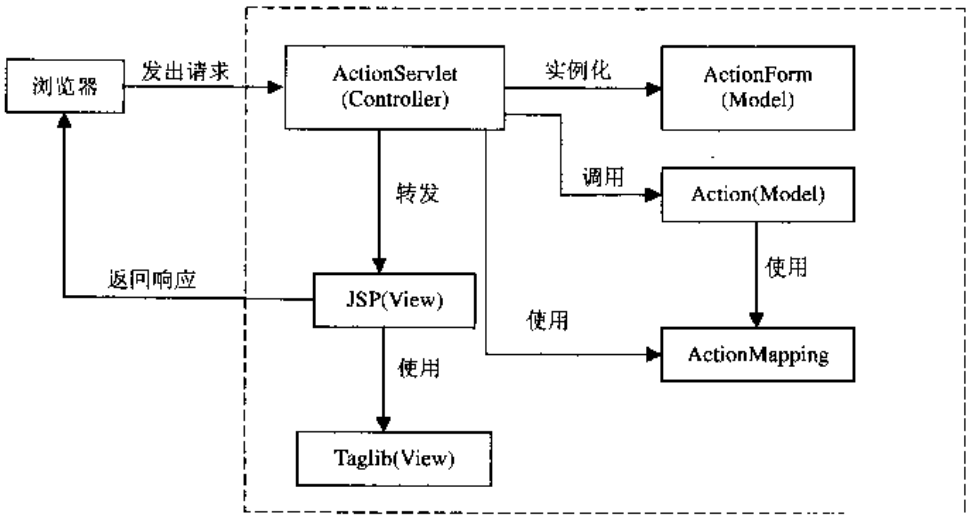


图 11-19 Struts 中和 MVC 模式相关的组件

1. 模型（Model）

在 Struts 框架中，ActionForm 和 Action 共同构成了 MVC 的模型（Model）类。

- 在 Struts 中 ActionForm 实际上就是一个 JavaBean，它是由 ActionServlet 创建并实例化后，以参数的形式传递到 Action 类。它既可以保存用户从视图（View）中输入的数据，也可以保存特定条件下从数据源（比如数据库）中取得的数据。对于从用户视图（View）中取得的数据，是在 bean 被实例化后，由 ActionServlet 自动为其设置的。如果 bean 中有 validate()方法，并且被要求进行数据校验，那么这个方法会先于 Action 类被调用执行，用来检测 bean 中属性的有效性。
- Action 类表示业务逻辑，由应用程序开发者对其进行扩展后开发的 Action 类，主要负责根据用户请求完成特定的业务逻辑。

2．视图（View）

由 JSP 利用 Struts 的标签库创建。

3．控制（Controller）

在 Struts 中控制是由 ActionServlet 和 ActionMapping 完成的。它们将用户端浏览器发出的请求封装，并转发到相应的对象（JSP 或者一个 Action 类）。

11.5.3　Struts 的配置文件

Struts 的配置文件是用来将 Struts 组件组合起来并表明组件之间关系的文件，是 Struts 中最重要的部分。前面说过在 Struts 中 ActionServlet 和 ActionMappinmg 是组成控制的主要类，负责封装和转发用户请求。ActionMapping 对象中保存的是请求和处理请求的组件之间的映射。

配置文件的名字和位置是在启动时由 ActionServlet 的初始化参数定义的。若程序开发者没有对其进行指定，那么默认的文件是/WEB-INF/struts-config.xml。

配置文件主要用于定义下列信息：

- 全局转发。主要用于在不同的 JSP 文件之间建立逻辑映射。全局转发在元素<forward>中被定义，而元素<forward>又被放置于<global-forwards>元素中。
- ActionForm bean。主要负责保存用户请求中的参数。ActionForm bean 在元素<form-bean>中被定义，而元素<form-bean>又被放置于<form-beans>元素中。
- ActionMapping。主要负责在请求 URI 和 Action 类之间建立映射。ActionMapping 在元素<action>中被定义，而元素<action>又被放置于<action-mappings>元素中。
- JDBC 数据源。在配置文件中还可以配置与外部数据源的连接。数据源在元素<data-source>中被定义，而元素<data-source>又被放置于<data-sources>元素中。

下面是一个简单的配置文件的例子。

```
    <?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 1.1//EN""http://jakarta.apache.org/struts/dtds/struts-
config_1_1.dtd">
    <struts-config>
        <form-beans>
            <form-bean name="stuffForm" type="test.example.StuffForm"/>
        </form-beans>

        <action-mappings>
            <action  path="/addStuff"
                type="test.example.addStuffAction"
                name="stuffForm"
                scope="request"
                validate="true"
                input="/add.jsp">
            <forward name="success" path="/finish.jsp"/>
            <forward name="failure" path="/add.jsp"/>
            </action>
        </action-mappings>
```

```
    <global-forwards>
        <forward   name="login"  path="/login.jsp"/>
        <forward   name="error"  path="/error.jsp"/>
    </global-forwards>

    <data-sources>
        <data-source
            key="myPool"
            driverClass="org.postgresql.Driver"
            autoCommit="true"
            description="My Test Data Source Configuration"
            maxCount="10"
            minCount="2"
            url="jdbc:postgresql://localhost:9090/testdata"
            user="struts"
            password="strutsPWD" />
    </data-sources>
</struts-config>
```

以这个文件为例，一起来看看配置文件中元素的属性和用法。

1. 配置全局转发

全局转发用来在 JSP 页之间创建逻辑名称映射。每一个全局转发都被单独保存在一个<forward>元素中，所有的<forward>元素都被放置在<global-forwards>元素中。可以通过调用 ActionMapping 的 findForward()方法以及<forward>元素的名字属性来访问这些转发。例如：

```
<global-forwards>
        <forward   name="login"  path="/login.jsp"/>
        <forward   name="error"  path="/error.jsp"/>
</global-forwards>
```

<forward>元素的属性如表 11-1 所示。

表 11-1 <forward>元素的属性

属 性 名	描 述
Name	全局转发的名字
Path	到目标 URL 的相对路径
Redirect	使用 RequestDispatcher.forward()方法，还是 response.sendRedirect()方法

2. 配置 ActionForm bean

ActionServlet 使用 ActionForm 保存请求中的参数。控制器将请求中的参数传递到 ActionForm bean 的实例，ActionForm bean 实例将参数保存好后，控制器再将这个 ActionForm bean 实例传送到 Action 类。ActionForm bean 的属性名称与 HTTP 请求参数中的名称相对应。例如：

```
<form-beans>
  <form bean name="stuffForm" type="test.example.StuffForm"/>
</form-beans>
```

这是使用 stuffForm 在其关联的作用域内声明了一个 test.example.StuffForm 类型的 ActionForm bean。<form-bean>元素的属性如表 11-2 所示。

表 11-2　<form-bean>元素的属性

属 性 名	描　　述
className	由 Struts 创建 ActionForm bean 时的完全限定名，默认值是 org.apache.struts.action.ActionFormBean，可以定义自己的 bean 类
name	在关联的作用域内的 form bean 的名称（也就是 request 或 session 对象保存这个 bean 时的属性名），通过这个属性 Struts 将 bean 与 ActionMapping 进行关联
type	类的完全限定名（包括包名）

3. 配置 ActionMapping

ActionMapping 的作用是将一个 URI 请求映射到某个指定的 Action 类上，以辅助 Struts 进行业务流程上的控制。通常 ActionMapping 指定的 Action 类名为完全限定名。ActionServlet 在内部保存这个映射，当客户端发出请求的时候，通过对请求的判断，ActionServlet 将请求转发给某个特定的 Action 类的实例。Action 类返回一个 ActionForward 对象，其中包括响应转发的目标资源名称。例如：

```
<action-mappings>
        <action path="/addStuff"
                type="test.example.addStuffAction"
                name="stuffForm"
                scope="request"
                validate="true"
                input="/add.jsp">
                <forward name="success" path="/finish.jsp"/>
                <forward name="failure" path="/add.jsp"/>
        </action>
</action-mappings>
```

这个映射将请求 URI /addStuff 匹配到一个 test.example.addStuffAction 的 Action 类上。一个逻辑名字为 stuffForm 的 form bean 在 request 作用域中和这个映射相关联。在调用 Action 的方法之前，会先调用 ActionForm bean 的 validate()方法进行数据检验。如果数据检验失败，则返回到 add.jsp。

<action>元素的属性如表 11-3 所示。

表 11-3　<action>元素的属性

属 性 名	描　　述
path	Action 类的相对路径
name	与 Action 关联的 Action bean 的名称，bean 的名称应该与<ActionForm bean>中定义的 bean 的名字相同
type	与这个映射相关的 Action 类的名称
scope	指明 ActionForm bean 的作用域，值为 request 或者 session

（续表）

属 性 名	描 述
prefix	用于匹配 request 参数与 bean 属性的前缀
suffix	用来匹配 request 参数与 bean 属性的后缀
attribute	保存 ActionForm bean 的作用域的名称
className	由 Struts 创建的 ActionMapping 对象的类的完全限定名。默认的类是 org.apache.struts.action.ActionMapping。如果需要，可以定义自己的类
input	输入表单的路径，指向当 bean 发生输入错误的时候，要返回的表单的路径
unknown	如果这个属性的值设为 true，那么这个映射将被作为所有没有被匹配的请求 URI 的默认映射
validate	如果这个属性的值设为 true，则在调用 Action 对象实例的 execute()方法前，ActionServlet 将调用 ActionForm bean 的 validate()方法进行输入的合法性检查

<action>元素中有一个子元素是<forward>。<forward>子元素是用来配置局部转发的元素。通过<forward>子元素可以定义目标资源的逻辑名称，该资源是 Action 类的响应要转发的目标。

<forward>元素的属性如表 11-4 所示。

表 11-4 <forward>元素的属性

属 性 名	描 述
className	ActionForward 类的完全限定名，默认值是 org.apache.struts.action.ActionForward。如果需要，可以定义自己的类
name	Action 类通过这个属性来访问 ActionForward。也就是 ActionForward 的逻辑名
path	用来响应转发的目标资源的路径
redirect	如果这个属性的值设为 true，则 ActionServlet 使用 sendRedirect()方法来转发资源，默认情况是使用 forward()方法来转发

4. 配置 JDBC 数据源

JDBC 数据源由<data-source>元素定义，在元素<data-sources>下可以定义多个不同的<data-source>元素，也就是说在 Struts 中可以配置多个不同的数据源。例如：

```
<data-sources>
        <data-source
        key="myPool"
        driverClass="org.postgresql.Driver"
        autoCommit="true"
        description="My Test Data Source Configuration"
        maxCount="10"
        minCount="2"
        url="jdbc:postgresql://localhost:9090/testdata"
        user="struts"
        password="strutsPWD" />
</data-sources>
```

在 Action 类中，如果想访问 JDBC 数据源，可以通过指定的 key 来访问。下面的代

码是在 Action 类中访问数据源的具体实现：

```
javax.sql.DataSource ds = servlet.findDataSource("myPool");
javax.sql.Connection con = ds.getConnection();
```

<data-source>元素的属性如表 11-5 所示。

表 11-5　<data-source>元素的属性

属 性 名	描　　述
key	Action 类使用这个名称来寻找连接
type	实现 JDBC 扩展 API 接口的类的名称
driverClass	被数据源所使用的类，用来实现 JDBC 驱动程序接口
desciption	对 JDBC 数据源的描述
autoCommit	用数据源创建的连接向数据库提交的模式，默认为自动提交
loginTimeout	数据库注册的时间限制，以秒为单位
maxCount	最多能建立的连接数目
minCount	要创建的最少连接数目
password	数据库访问的密码
readOnly	创建只读的连接
user	访问数据库的用户名
url	JDBC 的 URL

11.6　Struts 标签库

JSP 文件（View 组件）所使用的 Struts 标签库由六组标签组成。

1. HTML Tags

Struts 的 HTML 标签库包含用于生成 Struts 的用户界面的标签，该标签功能和 HTML 中的 FORM 相似，不同的是该标签将用户输入参数自动存储到指定的 ActionForm Bean 当中。它的标签库描述器文件是 struts-html.tld。

2. Logic Tags

Struts 的 logic 标签库包含用于控制输出文本条件、对集合对象迭代及应用流程控制方面的标签，该标签中的大部分标签功能在 JSTL 的 core 标签库中有相同或类似功能的标签。它的标签库描述器文件是 struts-logic.tld。

3. Bean Tags

Struts 的 Bean 标签库中的 JSP 自定义标签的主要作用是用于访问 Bean 和 Bean 的属性，以及基于这些访问而定义新的 Bean，使得页面可以通过脚本变量和页面内的属性来访问这些 Bean。它的标签库描述器文件是 struts-bean.tld。

4. Nested Tags

Struts 的 nested 标签库的作用是扩展其他标签库的标签，该标签库使得 Struts 标签库中的其它标签库可以被嵌套使用。它的标签库描述器文件是 struts-nested.tld。

5. Template Tags

Template 标签库包含的标签可用作为页面创建动态的 JSP 模板，这些页面都拥有一个公共的外观或者共同的格式。它的标签库描述器文件是 struts-template.tld。

6. Tiles Tags

Tiles 标签库提供一种模板机制，它能将布局和内容的职责分离。它的标签库描述器文件是 struts-tiles.tld。

下面介绍三组基础的标签：html 标签、logic 标签和 bean 标签。其他更多的信息请参考 Struts 自带的用户手册及文档。

11.6.1 HTML 标签

一个动态的 Web 应用程序需要从用户处获取数据，所以在 Struts 框架中输入表单扮演着重要的角色。而 html 标签库就是 JSP（View 组件）和其他组件之间的桥梁。因此，html 标签中的绝大多数都是 html 的表单对象。html 标签中其他重要的内容还有消息和国际化等。

下面介绍 html 标签库中的主要标签及其常用属性。

1. html 表单标签

<html:form>标签用来输出标准的 HTML 对象，可以指定 AcitonForm bean 的名称和它的类名。表单中的每一个对象都应该与 AcitonForm 中的属性对应。如果设置了这些属性，当表单被提交到 ActionServlet 的时候，用户输入的信息会被自动保存到 ActionForm 中。如果没有设置这些属性，就需要有配置文件来指定 ActionMapping，以表明当前输入的是哪个 JSP 页，以及从映射中检索的 bean 名和类。如果在 ActionMapping 所指定的作用域中没有找到指定的名称，就会创建并存储一个新的 bean，否则将使用找到的 bean。

<form>标签能够包含与各种 HTML 输入字段相对应的子标签。<html:form>标签属性有如下几种。

（1）action 属性：与表单相关的操作。在配置中，这个操作也用来标识与表单相关的 ActionForm。

（2）enctype 属性：表单 HTTP 方法的编码类型。

（3）focus 属性：表单中需要初始化焦点的对象。

（4）method 属性：表单使用的 HTTP 方法。

（5）name 属性：与表单相关的 ActionForm bean 的名称。如果没有设置这个属性，bean 的名称将会从配置信息中获得。

（6）onreset 属性：表单复位时的 JavaScript 事件句柄。

（7）onsubmit 属性：表单提交时的 JavaScript 事件句柄。

（8）scope 属性：搜索 ActionForm 的作用域。如果没有设置，将从配置文件中获取。

（9）style 属性：Form 所使用的格式。

（10）styleClass 属性：这个元素的格式表类。

（11）type 属性：ActionForm bean 的完整名称。如果没有设置，将从配置文件获得。

例句：

```
<html:form action="login.do" method="post">
</html:form>
```

说明：

与表单相关的操作路径是 login，而表单数据是通过 post 方式传递的。

对于这个表单来说，ActionForm bean 的其他信息，如 bean 名称类型，作用域等都是从表单指定操作的 ActionMapping 中得到的。在配置文件中将会有如下的信息：

```
<form-beans>
<form-bean name="loginForm" type="org.com.myExample.LoginForm"/>
<action-mappings>
    <action path="/lgoin"
            type="org.com.myExample.LoginAction"
            name=" loginForm"
            scope="request"
            input="/login.jsp">
        <forward name="success" path="/loginSuccess.jsp">
    </action>
</action-mapping>
```

上面这段代码所包含的信息是，login.jsp 文件中使用的 html 标签将使用“loginForm”作为它的名字属性，而这个属性的类型是“org.com.myExample.LoginForm”。如果在所有的作用域内（page，request，session，application）中都没有找到所指定的 ActionForm，那么，ActionServlet 将会在指定的作用域中，按照指定的类型创建一个 ActionForm 的实例并使用它。这些属性也可以使用<html:form>标签属性进行显示的设置，不过不赞成使用这种方式。

2．button 标签

<html:button>标签用于描绘一个按钮类型的<input>元素，必须被定义在<html:form>标签之中。属性如下。

（1）property 属性：定义在表单被提交时返回到服务器的请求参数的名称。

（2）value 属性：按钮上标签的内容。

（3）tabindex 属性：定义按钮的 tab 的顺序。

例句：

```
<html:button property="login" value="LOGIN" tabindex="1"/>
```

说明：

名为 login 的按钮，按钮上显示的是“LOGIN”，当用户单击 tab 键的时候焦点会首先聚焦在这个按钮上。

3. cancel 标签

<html:cancel>标签用于描绘一个提交类型的<input>元素，必须被定义在<html:form>标签之中。属性如下。

（1）property 属性：定义在表单被提交时返回到服务器的请求参数的名称。

（2）value 属性：按钮上标签的内容。

（3）tabindex 属性：定义按钮的 tab 的顺序。

例句：

```
<html:cancel property="cancel" value="CANCEL" tabindex="2"/>
```

说明：

名为 cancel 的按钮，按钮上显示的是“CANCEL”。

4. checkbox 标签和 multibox 标签

<html:checkbox>标签用于描绘一个类型为检查框的<input>元素。<html: multibox>标签用于描绘一个类型为复选框的<input>元素。它们都必须被定义在<html:form>标签之中。属性如下。

（1）name 属性：bean 的名称，其属性会被用来确定检查框是否以选中的状态显示。如果没有设置，将使用与这个内嵌表单相关的 ActionFrom bean 的名称。

（2）property 属性：检查框的名称，也是决定检查框是否以选中的状态显示的 bean 属性名称。在复选框的情况下，这个属性必须是一个数组。

（3）value 属性：当检查框被选中时返回到服务器的请求参数的值。

例句：

```
<html:checkbox property="sutdent" value="yes"/>
```

说明：

一个名为 sutdent 的检查框，在表单提交时会返回一个“yes”。

5. errors 标签

显示一个错误信息集合，由业务逻辑组件创建并存储在作用域中的 ActionErrors 对象中的错误集合。

<html:errors>标签的作用是显示一个由业务逻辑组件创建并存储在作用域中的错误信息集合（通常会被保存在 ActionErrors 对象中，或是一个 String 类型的数组中）。这个标签首先从当前作用域的资源文件中读取消息关键字 errors.header，然后显示消息的文本。它会在 ActionErrors 对象（通常作为请求参数而存储在 Action.ERROR_KEY 关键字下）中循环，读取单个 ActionError 对象的消息关键字，从当前作用域的资源文件中读取并格式化相应的消息，然后显示它们。通过定义 Property 属性能够过滤要显示的消息，这个属性的值应该与 ActionErrors 对象中存储 ActionError 对象的关键字对应。属性如下。

（1）bundle 属性：表示应用程序作用域属性的名称，它包含着消息资源，其默认值是 Acion.MESSAGE_KEY。

（2）locale 属性：表示会话作用域属性的名称，它存储着用户当前登录的区域信息。

其默认值是 Action.ERROR_KEY。

（3）name 属性：表示请求属性的名称，它存储着 ActionErrors 对象。其默认值是 Action.ERROR_KEY。

（4）property 属性：这个属性指定了 ActionErrors 对象中存储每个独立 ActionError 对象的关键字，它可以过滤消息。

例句：

```
<html:errors/>
```

说明：

显示 errors 集合中所有的错误。

例句：

```
<html:errors property="error1.name"/>
```

说明：

显示存储在 errors 集合中以 error1.name 为关键字的错误。

6. file 标签

<html:file>标签用于描绘一个文件类型的<input>元素，必须被定义在<html:form>标签之中。在使用这个标签的时候，相应的<html:from>标签的 method 属性应该被指定为“POST”，enctype 属性应该被指定为“multipart/form-data”，例如：

```
<html:form method="POST" enctype="multipart/form-data">
    <html:file property="theFile" />
</html:form>
```

file 标签的属性如下。

（1）name 属性：bean 的名称，它的属性将确定文件控件中显示的内容。如果没设置，将使用与内嵌表单相关的 ActionForm 的名称。

（2）property 属性：这个属性定义了当表单被提交时送回到服务器的请求参数的名称，以及用来确定文件控件中显示内容的 ActionForm 属性名称。

（3）accept 属性：服务器能够处理的内容类型集。它也将对客户浏览器对话框中的可选文件类型进行过滤。

（4）value 属性：按钮上的标记，这个按钮能够在本地文件系统中浏览文件。

7. hidden 标签

<html:hidden>标签用于描绘一个隐藏类型的<input>元素，必须被定义在<html:form>标签之中。属性如下。

（1）name 属性：bean 的名称，其属性会被用来确定隐藏元素的当前值。如果没有设置，将使用与这个内嵌表单相关的 ActionFrom bean 的名称。

（2）property 属性：定义了当表单被提交时送回到服务器的请求参数的名称，以及用来确定隐藏元素当前值的 ActionForm 属性的名称。

（3）value 属性：用来初始化隐藏输入元素的值。

例句：

```
<html:hidden property="hidInfo"/>
```

说明：

在 ActionForm 中对应着属性名为 hidInfo 的隐藏类型的 HTML 对象。

8. select 标签

<html:select>标签用于描绘一个 HTML 的选择对象，必须被定义在<html:form>标签之中。如果 Multiple 属性被设置为“true”，那么在对应的 ActionFrom 中的属性应该是某个类型的数组对象。属性如下。

（1）multiple 属性：表明这个选择对象是否允许进行多选。

（2）name 属性：bean 的名称，它的属性确定了使用哪个 bean。如果没有设置，将使用与这个内嵌表单相关的 ActionFrom 的名称。

（3）property 属性：定义了当表单被提交时送回到服务器的请求参数的名称，以及用来确定哪个选项需要被选中的 bean 属性的名称。

（4）size 属性：能够同时显示的选项数目。

（5）value 属性：用来表明需要被选中的选项。

9. option 标签

<html:option>标签用来描绘 HTML 选项元素。它必须要嵌套在<html:select>标签中。属性如下。

Value 属性：当下拉框中某一项被选中时返回到服务器的请求参数的值。

例句：

```
<html:select property="year" size="1" value="2004">
   <html:option value="2003"/>
   <html:option value="2004"/>
   <html:option value="2005"/>
</html:select>
```

说明：

上面这段代码将在 JSP 页面上生成一个名为 year 的下拉框，其中有三个选项（2003，2004，2005），默认被选中的是 2004。

10. options 标签

<html:options>标签用来描绘一个 HTML 选项元素的集合。它必须要嵌套在<html:select>标签中。属性如下。

（1）collection 属性：bean 集合的名称，这个集合存储在指定的作用域中。选项的数目与集合中元素的数目相同。property 属性能够定义选项值所使用的 bean 属性，而 Labelproperty 属性定义选项标签所使用的 bean 的属性。

（2）labelName 属性：用来指定存储于某个作用域的 bean，这个 bean 是一个字符串的集合，能够定义<html:option>元素的标签。

（3）labelProperty 属性：与 collection 属性共同使用时，用来定义存储于指定的作

用域的 bean，这个 bean 将返回一个字符串集合，能够用来写入<html:option>元素的 Value 属性。

（4）name 属性：如果这是惟一被指定的属性，它就定义了存储于指定的作用域的 bean，这个 bean 将返回一个字符串集合，能够用来写入<html:option>元素的 value 属性。

（5）property 属性：这个属性在与 collection 属性共同使用时，定义了每个要显示选项值的独立 bean 的 name 属性。如果不是与 collection 属性共同使用，这个属性定义了由 name 属性指定的 bean 的属性名称，或是定义了一个 ActionForm bean，这个 bean 将返回一个集合来写入选项的值。

例句：

```
    <html:option collection ="optionCollection" property ="optionValue"
labelProperty ="optionLabel"/>
```

说明：

上面的代码片断说明在某个作用域中有一个名为 optionCollection 的集合。它包含了一些具有 optionValue 属性的独立的 bean，每个属性将作为一个选项的值。每个选项的标志由 bean 的 optionLabel 属性进行定义。

例句：

```
    <html:option name="optionValues" labelName="optionLabels"/>
```

说明：

上面的代码片断的意思是 optionValues 代表一个存储在某个作用域中的 bean。它是一个字符串集合，能够用来写入 HTML 选项对象的值，而 optionLabels 代表存储在某个作用域中的另一个 bean。它也是一个字符串集合，能够用来写入 HTML 选项对象的标签，也就是显示的内容。

11．password 标签

<html:password>标签用于描绘 HTML 密码对象，必须被定义在<html:form>标签之中。属性如下。

（1）maxlength 属性：能够输入的最多字符数。

（2）name 属性：bean 的名称，其属性用来确定密码元素的当前值。如果没有设置，将使用与这个内嵌表单相关的 ActionFrom 名称。

（3）property 属性：定义了当表单被提交时送回到服务器的请求参数的名称，以及用来确定密码元素当前值的 bean 属性的名称。

（4）redisplay 属性：显示这个字段时，如果相应的 bean 属性已经被设置了数据，这个属性决定了是否显示密码的内容。

（5）size 属性：显示在页面上的长度。

12．radio 标签

<html:radio>标签用来描绘 HTML 单选按钮对象，必须被定义在<html:form>标签之中。属性如下。

（1）name 属性：bean 的名称，其属性被用来确定单选钮是否以选中的状态显示。如果没有设置，将使用与这个内嵌表单相关的 ActionFrom 的名称。

（2）property 属性：当表单被提交时送回到服务器的请求参数的名称，以及用来确定单选钮是否以被选中状态进行显示的 bean 属性的名称。

（3）value 属性：当单选钮被选中时返回到服务器的值。

13. reset 标签和 submit 标签

<html:reset>和<html:submit>标签分别用于描绘 HTML 复位按钮和提交按钮，他们都必须被定义在<html:form>标签之中。

14. text 标签和文本区标签

<html:text>和<html:textarea>标签分别用于描绘 HTML 文本框和文本区对象，它们都必须被定义在<html:form>标签之中。属性如下。

（1）property 属性：定义当表单被提交时送回到服务器的请求参数的名称，或用来确定文本元素当前值的 bean 的属性名称。

（2）name 属性：属性被查询的 bean 的名称，它决定了文本框和文本区的值。如果没有设置，将使用与这个内嵌表单相关的 ActionForm 的名称。

11.6.2 Logic 标签

Logic 标签库的标签被用来处理外观逻辑。Struts 的 Logic 标签库包含的标记能够有条件地产生输出文本，也可以在对象集合中循环从而重复地产生输出文本。它还提供了一组在 JSP 页中处理流程的控制标记。例如，对标签体或条件赋值。这些标记封装在文件名为 struts-logic.tld 的标记包中。按照功能 Logic 标签可以分为四部分。

1. 值的比较

这类标签有如下六个：

（1）<logic:equal>：如果常数与被定义的实体相等，返回 true。

（2）<logic:notEqual>：如果常数与被定义的实体不相等，返回 true。

（3）<logic:greaterEqual>：如果常数大于等于被定义的实体，返回 true。

（4）<logic:lessEqual>：如果常数小于等于被定义的实体，返回 true。

（5）<logic:lessThan>：如果常数小于被定义的实体，返回 true。

（6）<logic:greaterThan>：如果常数大于被定义的实体，返回 true。

它们共同的属性如下所示：

（1）value 属性：要进行比较的常数值。

（2）cookie 属性：要进行比较的 HTTP Cookie 的名称。

（3）header 属性：要进行比较的 HTTP 请求标头的名称。

（4）parameter 属性：要进行比较的 HTTP 请求参数的名称。

（5）name 属性：如果要进行比较的是 bean 或 bean 的属性，则这个属性代表 bean 的名称。

（6）property 属性：要进行比较的 bean 属性的名称。

（7）scope 属性：bean 的作用域，如果没有指定作用域，则它的搜索范围是从页作用域到应用程序作用域。

例句：

```
<logic:equal parameter="name" value="Tom">
    The entered name is Tom !
</logic:equal>
```

说明：

判断用户请求中名为“name”的参数的值是否是“Tom”。如果结果为真，则显示“The entered name is Tom!”。

例句：

```
<logic:greaterThan name="personBean" property="age" scope="page" value="18">
    The age of the person is greater than 18!
</logic:greaterThan>
```

说明：

判断在页的作用域中是否有一个名为“personBean”的 bean，它有一个 age 属性，这个属性的值是否大于 18。如果这个属性能够转化为数值，则按照数值进行比较，否则按照字符串对象进行比较。如果结果为真，则显示“The age of the person is greater than 18!”。

2. 字符串匹配

这类标签有如下两个。

（1）<logic:match>：如果一个指定项目的值是一个特定常数的子字符串，返回 true。

（2）<logic:notMatch>：如果一个指定项目的值不是一个特定常数的子字符串，返回 true。

它们的属性如下。

（1）cookie 属性：要进行比较的 HTTP Cookie 的名称。

（2）header 属性：要进行比较的 HTTP 标头的名称。

（3）parameter 属性：要进行比较的 HTTP 请求参数的名称。

（4）name 属性：若要对 bean 或 bean 的属性进行比较，这个属性是用户指定 bean 的名称。

（5）location 属性：有“start”和“end”两个可能的值。如果设置了这个属性的值，将会在这个指定的位置（开始或结尾处）进行匹配。

（6）scope 属性：如果对 bean 进行比较，这个属性指定了 bean 的作用域。如果没有设置这个参数，搜索范围是从页作用域到应用程序作用域。

（7）property 属性：要进行比较的 bean 的属性名称。

（8）value 属性：要进行比较的常数值。

例句：

```
<logic:match parameter="name" value="eclipse" location="start">
```

```
        The parameter name is a string start with the string eclipse。
    </logic:match>
```

说明：

标签检查用户请求中名为“name”的参数是否是以“eclipse”开头的子字符串。

3. 转发和重定向标签

（1）转发标签：<logic:forward>标签能够将响应转发给特定的 ActionForward。ActionForward 的类型决定了是使用 PageContext 转发响应，还是使用 sendRedirect 将响应进行重定向。此标记只有一个“name”属性，用来指定全局 ActionForward 的名称。

例子：

```
<logic:forward name="myActionForward"/>
```

（2）重定向标签：<logic:redirect>标签是一个能够执行 HTTP 重定向的工具。根据指定的不同属性，它能够通过不同的方式实现重定向。属性如下：

① forward 属性，映射了资源相对路径的 ActionForward。

② href 属性，资源的完整 URL。

③ page 属性，资源的相对路径。

④ name 属性，使用一个 Map 保存用户请求中参数的 bean 的名字（没有设置 property 属性的情况），或者是类型为 Map 的 page，request，session 或 application 属性的名称，它包含了相应的重定向的 URL 的查询参数的“名-值”对（设置了 property 属性的情况）。

⑤ property 属性：Map 类型的 bean 属性的名称。bean 的名称由 name 属性指定。若指定 Property 属性，那么必须指定 Name 属性。

⑥ scope 属性，如果指定了 bean 的名称，这个属性指定搜索 bean 的作用域。如果没有设置，则搜索范围从页作用域到应用程序作用域。

⑦ paramID 属性，定义特定查询参数的名称。

⑧ paramName 属性，字符串类型的 bean 的名称，其中包含查询参数的值（如果没有设置 paramproperty 属性）。或是一个 bean 的名称，它的属性（在 paramproperty 属性中指定）包含了查询参数值。

⑨ paramproperty 属性，字符串类型的 bean 属性的名称，其中包含着查询参数的值。

⑩ paramScope 属性，paramName 定义的 bean 的搜索范围。

使用这个标记时至少要指定 forward, href 或 page 中的一个属性，以便标明将响应重定向到哪个资源。

例子：

```
<%request.setAttribute("userID","Tom");%>
<logic:redirect hert=http://localhost:8080/example
    paramId="userID"
    paramName="userID"
    paramScope="request"/>
```

例句说明：

上面代码片断将响应重定向到 http://localhost:8080/example?userID=Tom。

4．循环标签

在逻辑标记库中定义了<logic:iterate>标记，它能够根据指定集合中元素的数目对标签体的内容进行重复的执行。集合的类型可以是一个迭代器（java.util.Iterator）、一个集合（java.util.Collection 包括 ArrayList 和 Vector）、一个枚举类型（java.util.Enumeration）、一个 Map（java.util.Map 包括 HashMap,Hashtable 和 TreeMap）或是一个数组。用来被循环的集合必须采用下列方式之一定义：

（1）使用运行时间表达式来返回一个属性集合的集合。

（2）将集合定义为 bean，并且使用 Name 属性指定存储属性的名称。

（3）使用 name 属性定义一个 bean，并且使用 Property 属性定义一个返回集合的 bean 属性。

当前元素的集合会被定义为一个“页作用域”的 bean，属性如下。

（1）collection 属性：如果没有设置 name 属性，它就指定了要进行重复的集合。

（2）id 属性：页作用域 bean 和脚本变量的名称，它保存着集合中当前元素的句柄。

（3）indexed 属性：页作用域 JSP bean 的名称，它包含着每次重复完成后集合的当前索引。

（4）length 属性：重复的最大次数。

（5）name 属性：作为集合的 bean 的名称或是单个 bean 的名称。如果是单个 bean 的名称，则由 property 属性定义的 bean 属性的类型应该是个集合。

（6）offset 属性：重复开始位置的索引。

（7）property 属性：作为集合的 bean 属性的名称。

（8）scope 属性：如果指定了 bean 名称，这个属性设置 bean 的作用域。若没有设置，则搜索范围从页作用域到应用程序作用域。

（9）type 属性：当前定义的页作用域 bean 的类型。

11.6.3　Bean 标签

Bean 标签库除了可以在输出响应中实现特定的 bean 或 bean 属性之外，还可以通过各种可能的数据源在作用域内定义新的 bean。

Bean 标签库所包含的标签可以通过脚本变量和页面的 Scope 属性建立一个新的 bean 来保存页面的信息（如果需要的话），还可以对 bean 及其属性进行访问。而且还提供了通过所接受到的用户请求中的 Cookies，Headers 和 Parameters 很方便地建立新的 bean 的机制。在标准 JSP 文件中，由<jsp:useBean>创建 bean，而 bean 标签库扩展了<jsp:useBean>所提供的基本功能。bean 标签库中的许多标签，当没有被正确的利用的时候，会在运行的时候抛出 JspException 异常。如果希望对抛出的异常进行处理，JSP 文件允许用户声明一个 error page。

1．define 标签

<bean:define>定义一个基于指定 bean 属性的脚本变量，属性如下。

（1）body 属性：定义这个标签的内容。

（2）id 属性：定义将被显示在 page 作用域内的 bean 脚本变量名称。

（3）name 属性：用于显示 bean 的名称。若 Value 属性没有设置，这个属性必须设置。

（4）type 属性：定义引入脚本变量的类。

（5）value 属性：为 id 属性定义的脚本变量分配一个新的对象。

（6）property 属性：name 属性定义的 bean 的属性名称，用来定义新的 bean。

（7）scope 属性：源 bean 的作用域。若没有设置，搜索范围从页作用域到应用程序作用域。

（8）toscope 属性：目标 bean 的作用域。若没有设置，默认值是页作用域。

<bean:define>可以实现 bean 的复制。

例句：

```
    <bean:define id="targetBean" name="sourceBean"scope="page"
toScope="request"/>
```

说明：

一个名为“sourceBean”的 bean，作用域为 page，把它复制到 request 的作用域中，名为 targetBean。

2. cookie,head 和 parameter 标签

<bean:cookie>，<bean:head>和<bean:cookie>这三个标签用于从 cookie，HTTP 头和用户请求中取值。它们共同的属性如下。

（1）id 属性：要定义脚本变量和的 page 作用域属性的名称。

（2）multiple 属性：如果存在多个相同名字的 cookie/head/parameter，是否定义一个 cookies/string 数组。如果没有设置这个属性，那么指定 cookie 的第一个值将作为 Cookie 类型的值。

（3）name 属性：cookie/head/parameter 的名称。

（4）value 属性：如果没有匹配的 cookie 或数值，就返回这个属性所指定的默认值。

<bean:cookie>标签的例子：

```
    <bean:cookie id="myCookie" name="userName"/>
```

说明：

脚本变量名称是 myCookie，用于创建这个属性的 cookie 的名称是 userName。

<bean:header>标签的例子：

```
    <bean:header id="myHeader" name="Accept-Language"/>
```

说明：

脚本变量名称是 myHeader，请求标头的名称是 Accept-Language。

<bean:parameter>标签的例子：

```
    <bean:parameter id="myParameter" name="myParameter">
```

说明：

脚本变量名称是 myPatameter，保存它的请求参数的名称也是 myParameter。

3. include 标签

<bean:include>标签将读取一个资源的响应。这个资源可以是 page，ActionForward 或外部 URL。它与<jsp:include>的不同点为资源的响应是被保存到一个 page 作用域的 bean 中，而不是写入到输出流，属性如下。

（1）id 属性：脚本变量和要定义的 page 作用域属性的名称。

（2）page 属性：一个内部资源。

（3）forward 属性：Struts 框架中的 ActionForward 类。

（4）href 属性：要包含的资源的完全限定 URL 地址。

例句：

```
<bean:include id="myInclude" href="http://localhost:8080/index"/>
```

说明：

脚本变量的名称是 myInclude，要读取的响应来自资源 http://localhost:8080/index。

4. resource 标签

<bean:resource>标签用于检索 Web 应用中的资源，并引入一个脚本变量和 InputStream 或字符串类型的 page 作用域属性，属性如下。

（1）id 属性：脚本变量的名字和要定义的 page 作用域属性。

（2）name 属性：到资源的相对路径。

（3）input 属性：如果属性不存在，资源可以作为字符串被访问。

例句：

```
<bean:resource id="myResource" name="/WEB-INF/myResource.recource"/>
```

说明：

脚本变量的名称是 myResource，要检索的资源名称是 myResource.recource。

5. write 标签

<bean:write>主要用于输出 bean 的属性，类似于标准的 JSP 中的<jsp:getProperty>。属性如下。

（1）name 属性：要进行属性显示的 bean 的名称。

（2）property 属性：要显示的属性的名称。如果这个属性类有 java.beans. Property Editor, getAsText()或 toString()方法，那么，这些方法会被调用。

（3）scope 属性：Bean 的作用域。若没有设置，搜索范围从页作用域到应用程序作用域。

（4）filter 属性：如果设置为 true，那么属性中的所有特殊 HTML 字符都将被转化为相应的实体引用。

（5）ignore 属性：如果设置为 false，那么当发现属性时会产生一个请求时间异常，否则返回 null。

例句：

```
    <bean:write name="myBean" property="myProperty" scope="request"
filter="true"/>
```

说明：

myBean 的属性 myProperty 将会被显示，作用域为请求，如果发现任何 HTML 特殊字符都将被转化为相应的实体引用。

11.7 Easy Struts 插件

Easy Struts 是 Eclipse 的一个用来开发 Struts 应用程序的插件，可以从地址：http://easystruts.sourceforge.net/下载 org.easystruts.eclipse_0.6.4.zip 的 Easy Struts 压缩文件。

Easy Struts 的安装步骤如下。

（1）将下载的压缩文件 org.easystruts.eclipse_0.6.4.zip 解压，把解压后的文件夹 com.cross.easystruts.eclipse_0.6.4 拷贝到 Eclipse 下面的 plugins 文件夹下面，如图 11-20 所示。



图 11-20 文件夹结构

（2）启动 Eclipse，通过菜单“窗口（W）”→“首选项（P）”打开设置窗口，如图 11-21 所示。

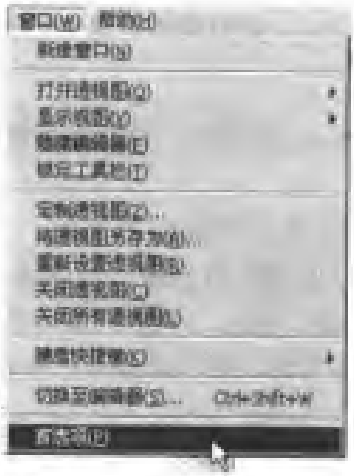


图 11-21 选择首选项

选择窗口的左侧的选项：Easy Struts，选择右侧的 Struts 1.1，单击“Add JARs...”

把从 Jakarta Struts 网站上下载的*.jar 添加进来。单击“Add TLDs...”，将 Struts 所需的类型库描述文件导入。其余的选项无需改动，保持默认值即可。单击“确定”，关闭窗口（如图 11-22 所示）。



图 11-22　Easy Struts 选项卡

至此已经完成 Easy Struts 的基本设置，可以开始 Struts 项目的开发了。

11.8　用 Easy Struts 插件开发 Struts 项目

在下面应用程序示例中，实现了 Web 应用程序中经常使用的功能——登录。这个流程很简单，如图 11-23 所示。

11.8.1　新建 Struts 工程

（1）新建一个 Tomcat 工程，名字叫“struts_Test02”（请参阅第 8 章），新建的工程结构如图 11-24 所示。

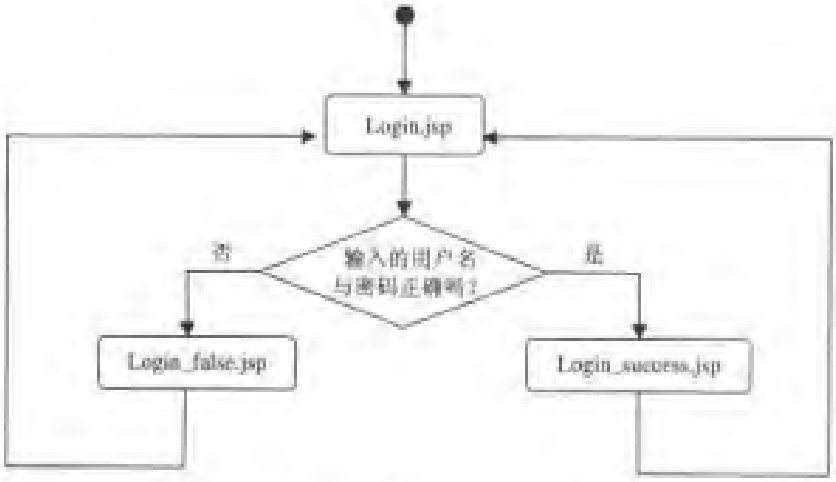


图 11-23　登录功能的流程



图 11-24　新建 Tomcat 工程

（2）集成 Easy Struts。

下面把 Easy Struts 集成到新建的工程上，在新的工程上右击鼠标，选择“新建”→“其它（O）”（如图 11-25 所示）。



图 11-25　新建 Easy Struts

在新弹出的界面中，选择左侧的“Easy Struts”，再选择右侧的“Add Easy Struts support”，如图 11-26 所示。

单击“下一步”，进入下一个界面，这里是用来填写 Easy Struts 的基本信息的，在 Webapp folder 中填写应用程序的目录名，Web application version 选择 2.3 和 Struts1.1。修改 Easy Struts 自动产生的程序包名，例如 test.struts。当修改“Base package for generation”的时候，“Default application resource”会自动的根据输入而改变（如图 11-27 所示）。

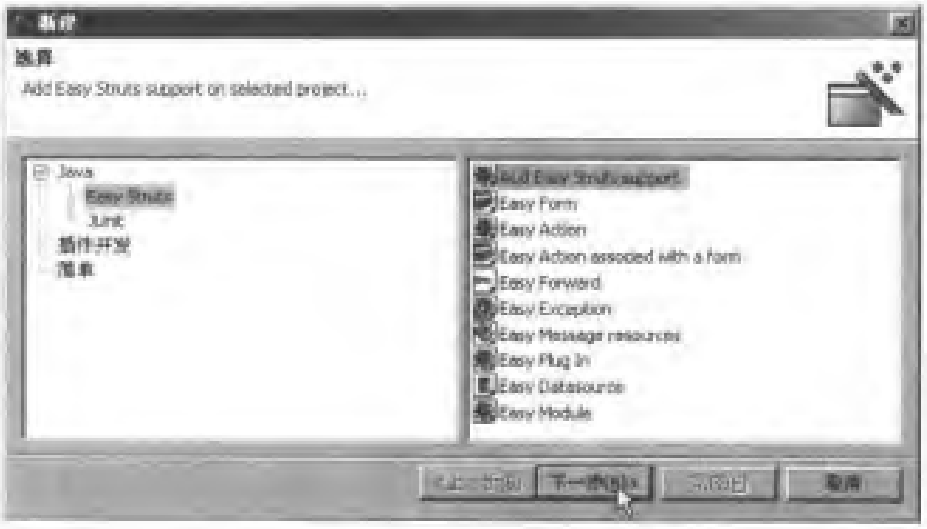


图 11-26　新建选项卡

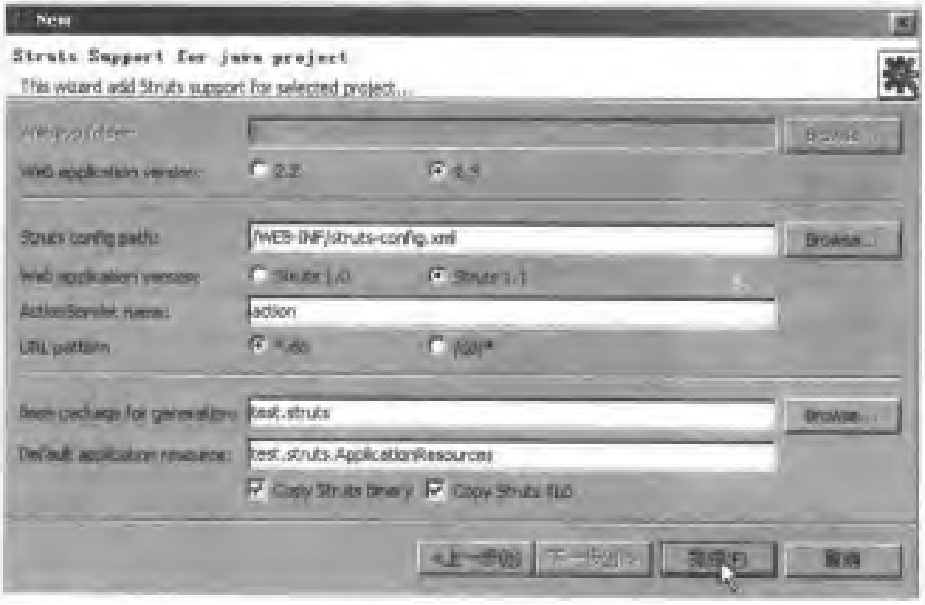


图 11-27　新工程的基本信息

单击“完成”按钮后，可以看到在创建的工程中发生了变化，/WEB-INF/下面多了一些文件，/WEB-INF/src/下面多了一个文件夹，/WEB-INF/lib/下面多了文件 struts.jar（如图 11-28 所示）。

这里对工程的设置，可以随时修改。在工程上右击鼠标，选择“属性”，在新弹出来的界面中，选择左侧的“Easy Strtuts”，可以看到右侧的选项卡中将对工程的设置都显示了出来（如图 11-29 所示）。

可以对已经设置好的属性进行修改。

现在工程建好了，Easy Struts 也已经集成到工程上，可以进行编码了。



图 11-28　导航器视图

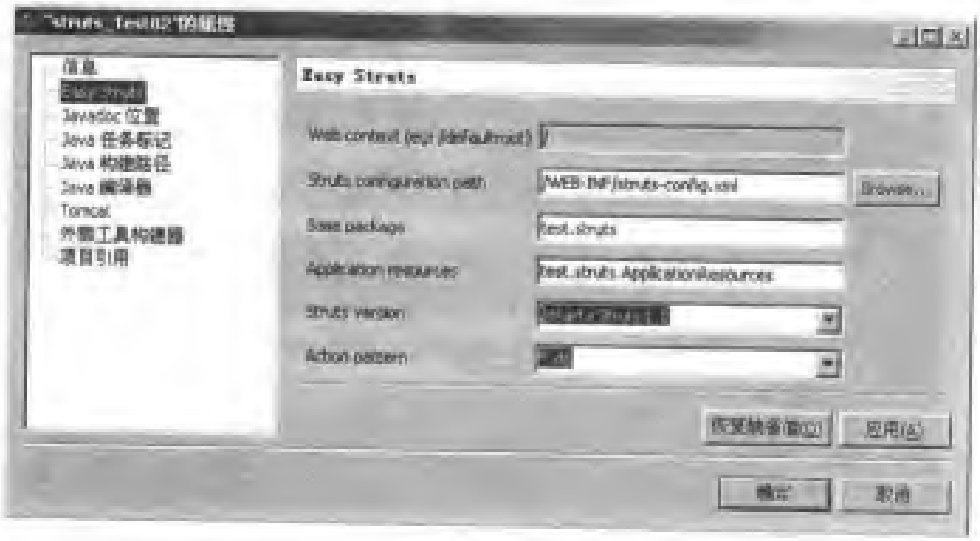


图 11-29　属性选项卡

11.8.2　生成 JSP 文件

首先从最简单的视图（View）开始创建。视图（View）是通过 JSP 实现的。从程序的流程图上可以看出共有 3 个 JSP 文件，login.jsp，login_success.jsp 和 login_false.jsp。下面首先生成这三个 JSP 文件。

在工程上单击右键，选择“新建”→“文件”，在新弹出的界面中选择生成文件的文件夹，填写文件的文件名“login.jsp”，用同样的步骤生成其余的两个 JSP 文件（如图 11-30 所示）。



图 11-30　新建 jsp 文件

三个 JSP 文件都做好后，可以看到在工程目录下生成了三个 JSP 文件（如图 11-31 所示）。



图 11-31　导航器视图

暂时不对这三个文件做修改。

11.8.3　生成 ActionForm 类

下面生成 ActionForm 类，在工程上单击右键，选择“新建”→“其它”，在新弹出的界面中，选择左侧的“Easy Struts”，再选择右侧的“Easy Form”→“下一步”（如图 11-32 所示）。

在下一页——“Use case”中填写 login。因为已经手动建立了 login.jsp，所以“（Create Jsp）input”这一项，不需要选中（Struts 默认的是选中的）。Struts 还会默认创建 validate() 和 reset()这两个方法。validate()方法是用来检查数据的合法性，reset()是把 bean 的属性恢复成默认值。因为例子很简单，所以暂时不需要创建这两个方法。

为 Form 添加属性，单击“Form properties”这一项目旁边的“Add”按钮。弹出如图 11-33 的界面，添加一个叫“s_Name”的属性，用来存储用户名。“JSP input type”这一项选择“text”。单击“确定”按钮，完成添加。



图 11-32　新建选项卡

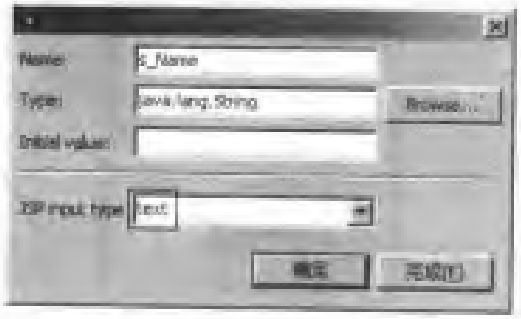


图 11-33　添加 s_Name 属性

再为这个Form添加第二个属性，名称叫“s_password”，用来存储用户密码，“JSP input type”这一项选择“password”（如图 11-34 所示）。单击“确定”按钮，完成添加。



图 11-34　添加 s_password 属性

添加完属性后（见图 11-35），单击“完成”。

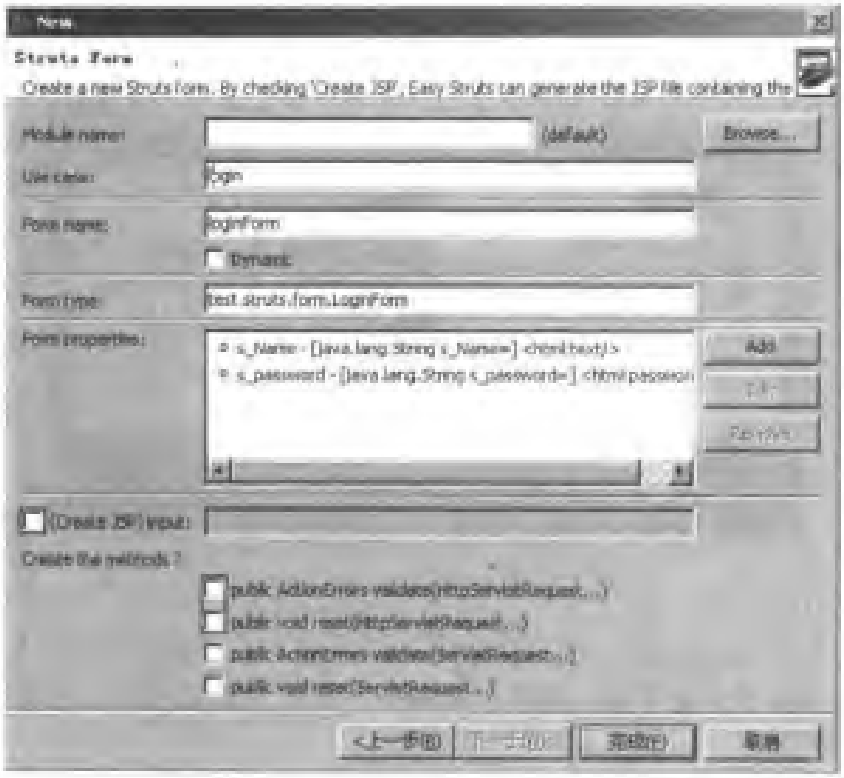


图 11-35　Struts Form 选项卡

可以看到，在/WEB-INF/src/test/struts/form/下面生成了一个文件 loginForm.java（如图 11-36 所示）。

接下来，该生成 Action 类了。

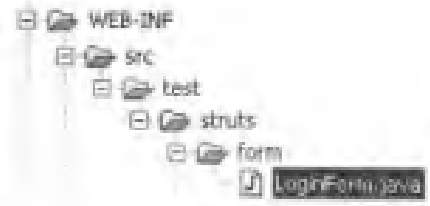


图 11-36　导航器视图

11.8.4　生成 Action 类

1. 生成 LoginAction 类

这个类用来检测用户所输入的用户名字和密码是否正确。如果输入正确，则打开 login_success.jsp 文件，如果输入错误，则打开 login_false.jsp 文件。

在工程上右击鼠标，选择“新建”→“其它”，在新弹出的界面中，选择左侧的“Easy Strtuts”，再选择右侧的“Easy Action”→“下一步”（如图 11-37 所示）。

在新弹出的界面的“Use case”中填写 login。

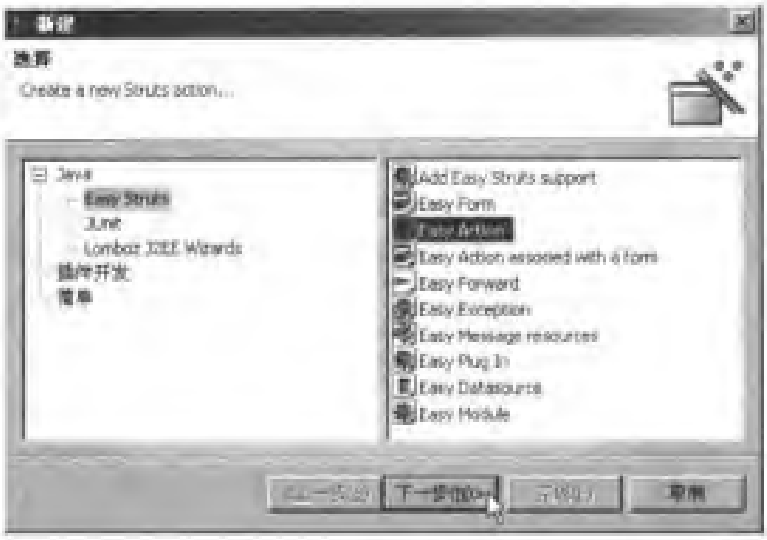


图 11-37　生成 Action 类

还要为这个 Action 添加 Form，单击“Form name”这一项目旁边的“Browse”按钮，弹出（如图 11-38）所示的界面，选择 loginForm。单击“确定”按钮，完成添加。其余默认的选项可以不修改。

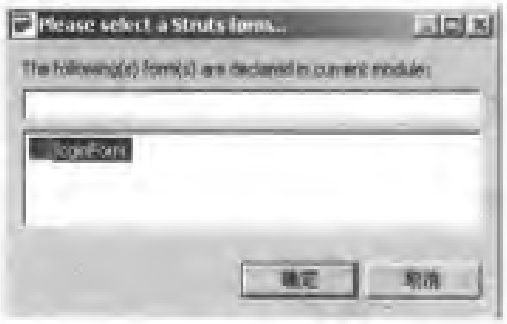


图 11-38　选择 loginForm

完成后的界面如图 11-39 所示。

单击“下一步”按钮，弹出新的窗体，它是用来添加目标 JSP 文件的。单击“Forwards”对应的“Add”按钮，在新弹出的窗体中——“Forward name”中填写“success”，在“Forward path”中填写“/login_success.jsp”（见图 11-40），也可以单击“Browse”按钮来指定 JSP 文件。该步的目的是要告诉 action servlet，当 LoginAction 返回“success”的时候，则打开/login_success.jsp 文件。单击“确定”，再继续添加下一个。

下一个 Forward 的名字是“false”，所对应的文件是“/login_false.jsp”（如图 11-41 所示）。

单击“完成”生成 LoginAction 类。

2. 生成 LoginoffAction 类

和生成 LoginAction 类的步骤一样，但有一点不同的是，不需要为 LoginoffAction 指定 ActionForm。（如图 11-42 所示。）

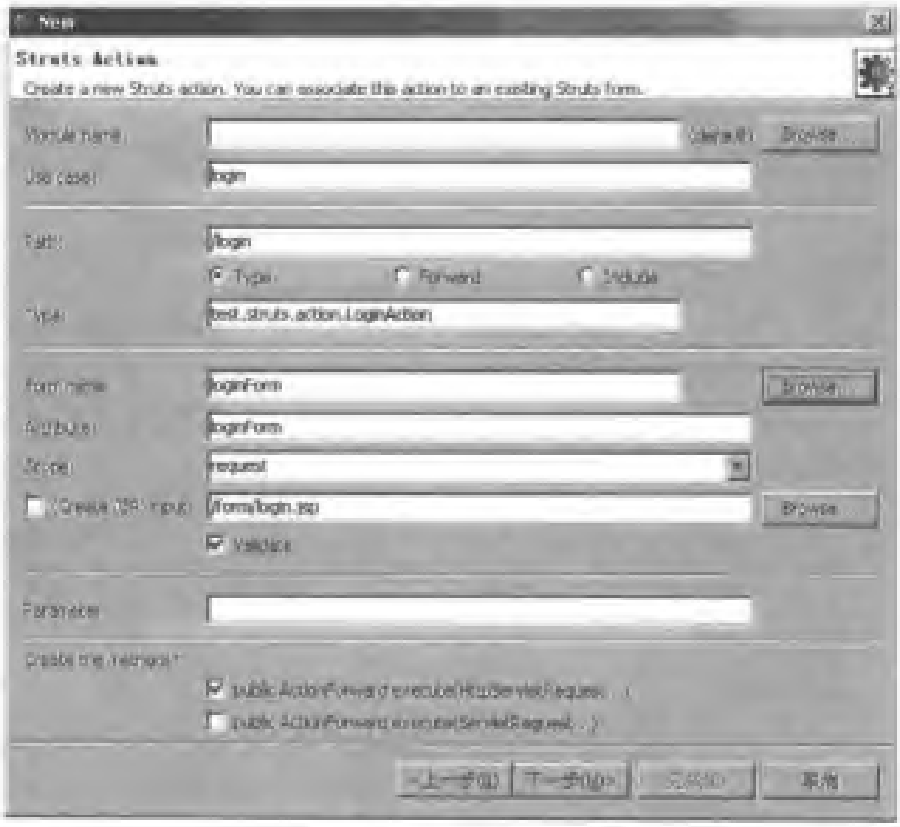


图 11-39　Struts Action 类选项卡



图 11-40　添加目标 JSP 文件



图 11-41　添加目标 JSP 文件

在单击“下一步”后弹出的新的界面中，也不需要做任何添加，单击“完成”即可。

可以从左面的导航器中看到，在/WEB-INF/src/test/struts 文件夹下面又生成了一个 action 文件夹。这个文件夹是系统自动生成的用来存放 Action 类的文件夹。刚才生成的 LoginAction 和 LogoffAction 就放在这个文件夹下面（如图 11-43 所示）。

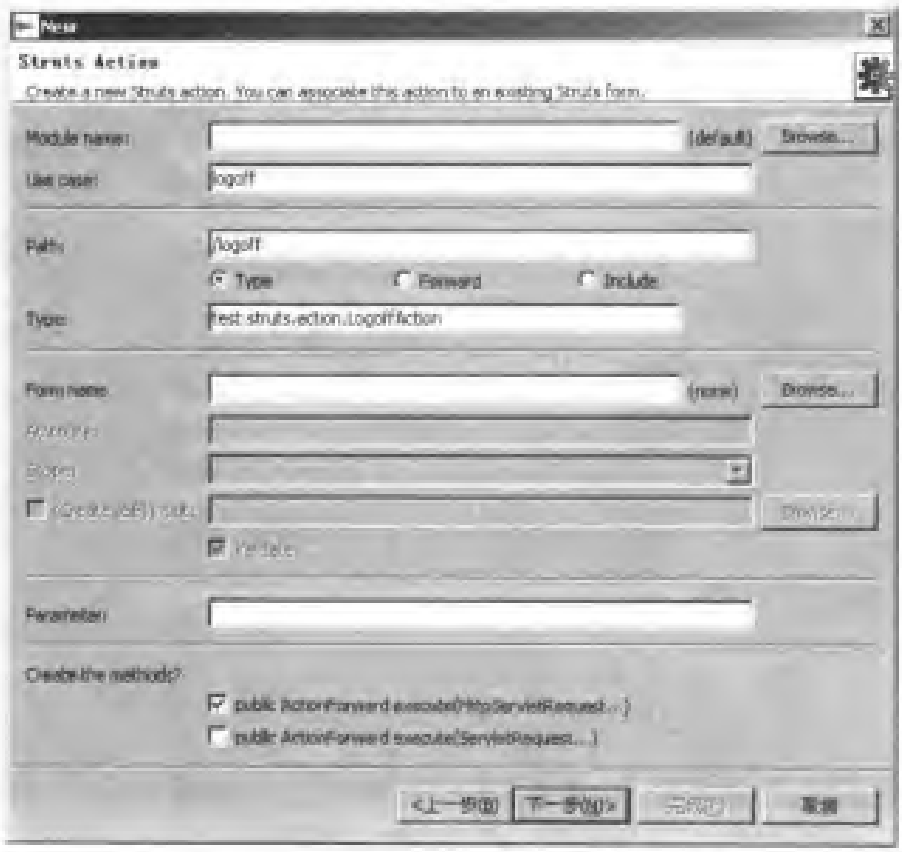


图 11-42　Struts Action 类选项卡



图 11-43　导航器视图

11.8.5　生成 test.properties 文件

在这个文件中来存放用户名和密码，用来和用户输入的用户名和密码进行比较。但是，在实际工作中，把用户名和密码放在这个文件中是不明智的（一般情况下这些信息都是加密后保存在数据库中，有关数据库的连接和访问请参考第 10 章）。在这里使用这个文件来模拟数据库。当然，这个文件可以放在任何地方，名字也可以是任意的。

文件中的内容如下：

```
Name=Tom
Password=struts
```

11.8.6　配置 struts-config.xml 文件

双击 Eclipse 左侧导航器中 WEB-INF 下面的 struts-config.xml，可以看到刚才创建的所有东西（如图 11-44 所示）。

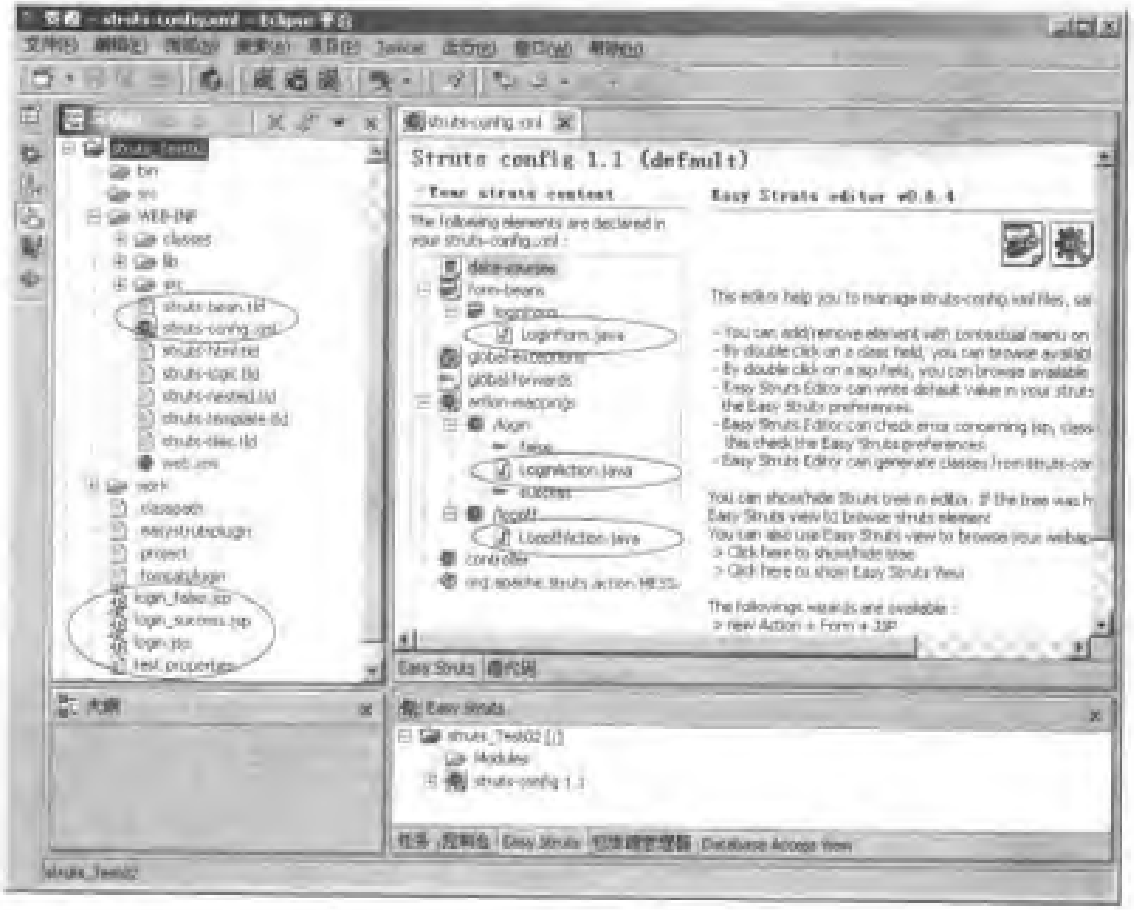


图 11-44　Eclipse 工作台

Easy Struts 提供了一个图形化的界面来管理 struts-config.xml（如图 11-45 所示），单击 loginForm，可以在右侧看到它所对应的类和它的属性。还可以通过双击 LoginForm.java 来打开这个类对它进行修改，同样还可以察看“/login”和“/logoff”等。

可以看到“/logoff”没有“forwards”。当 Struts 接收到它的返回值的时候，应该打开哪个文件呢？“global-forwards”就是用来解决这个问题的。下面来新建一个“global-forwards”。在 global-forwards 上右击鼠标→“New”→“Forward”，会看到在“global-forwards”下面生成了一个新的标签（如图 11-46 所示）。

在右侧的部分中填写这个新标签的名字和所对应的 JSP 文件。这个新建的“global-forwards”的意思是，不管是哪一个 Action 类，如果它返回的是“false”那么，

Struts 就会打开/login.jsp 文件。但是，需要注意的一点是一个 Action 的 Forward 和“global-forwards”的名字一样，那么 Struts 打开的是 Action 的 Forward 所对应的文件。对本例而言，Logoff 返回 false 的时候，Struts 会打开/login.jsp。而 Login 返回 false 的时候，Struts 会打开/login_false.jsp（如图 11-47 所示）。

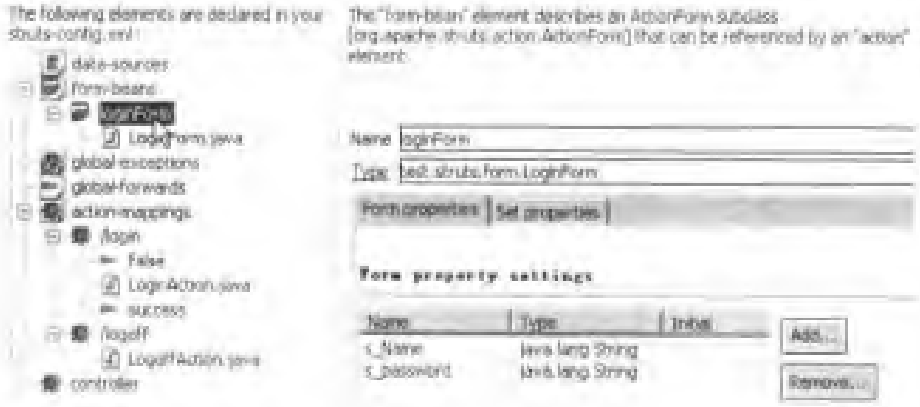


图 11-45 图形化的管理界面

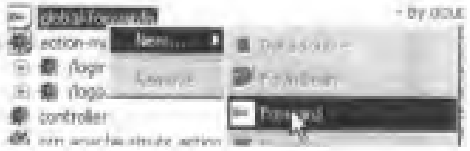


图 11-46 为/logoff 添加 forwards

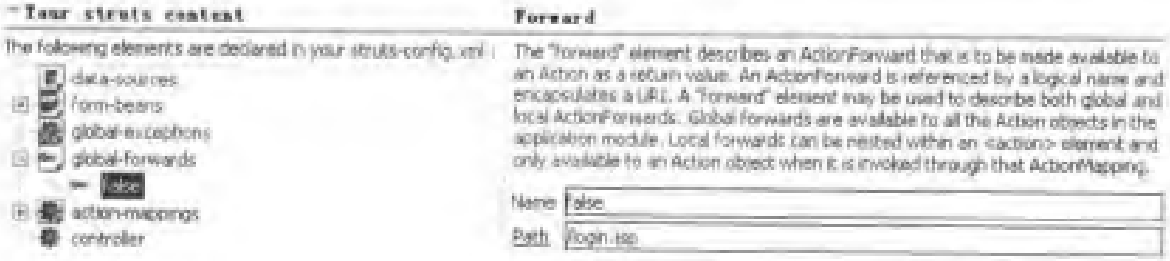


图 11-47 forwards 属性

11.8.7 生成数据库访问文件

还需要一个类用来模拟从数据库中取值。

在工程上右击鼠标，选择“新建”→“其它(O)”，在新弹出的界面中左面选择“Java”，右面选择“类”→“下一步”（如图 11-48 所示）。

在新的界面中填写“包(K)”名“test.struts.database”，和“名称(M)”“SelectData”，单击“完成”（如图 11-49 所示）。



图 11-48　新建类



图 11-49　填写包和类名

可以从左面的导航器中看到，在/WEB-INF/src/test/struts 文件夹下面又生成了一个 database 文件夹，这个文件夹下面存放的就是新做的 SelectData 类。

为这个类添加一个方法，来模拟验证用户身份的过程（如图 11-50 所示）。

这个方法的具体工作过程是，调用这个方法，并且把需要验证的用户名字和密码作为参数传入方法中，这个方法再从文件中（就是在 11.8.5 节中创建的 test.properties 这个文件）取得预先保存的用户名和密码，把他们进行比较，如果相同则返回“true”，如果不同，则返回“false”。

现在，主要的部件都已经做好了，将其修改后即可组装起来。

```
public boolean selectDataFromDB(String ps_name, String ps_pwd) {
    boolean b_return = false;
    String s_name = "";
    String s_PWD = "";

    try {
        FileInputStream f = new FileInputStream
        ("D:\\eclipse\\workspace\\struts_Test02\\test.properties");
        Properties property = new Properties();
        property.load(f);
        s_name = property.getProperty("Name");
        s_PWD = property.getProperty("Password");
    } catch (Exception e) {
        e.printStackTrace();
    }
    if (s_name == null || s_PWD == null) {
        b_return = false;
    } else {
        if (!s_name.equals(ps_name) || !s_PWD.equals(ps_pwd)) {
            b_return = false;
        } else {
            b_return = true;
        }
    }
    return b_return;
}
```

图 11-50　SelectData.java 文件

11.8.8　编辑文件

1. 编写 login.jsp 文件

下面编写一个 jsp 文件作为登录画面，具体程序代码如图 11-51 所示。

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<html>
<head><meta http-equiv="content-type" content="text/html; charset=gb2312">
<title><bean:message key="login.title"/></title></head>
<body bgcolor="#FFFFFF">
<html:form action="/login.do" focus="s_Name">
<table border=0>
  <tr>
    <th align="right"><bean:message key="prompt.userName"/></th>
    <td align="left">
      <html:text property="s_Name" size="16" maxlength="16"/>
    </td>
  </tr>
  <tr>
    <th align="right"><bean:message key="prompt.password"/></th>
    <td align="left">
      <html:password property="s_password" size="16" maxlength="16"/>
    </td>
  </tr>
  <tr>
    <td align="right">
      <html:submit>
        <bean:message key="button.submit"/>
      </html:submit>
    </td>
    <td align="left">
      <html:reset>
        <bean:message key="button.reset"/>
      </html:reset>
    </td>
  </tr>
</table>
</html:form>
</body>
</html>
```

图 11-51　login.jsp 文件

代码中类似<bean:message key="login.title"/>的部分，就是 Struts 的标签了。此标签的意思是输出/WEB-INF/src/test/struts/ApplicationResources.properties 这个文件中以 login.title 为键值的内容。因此，要向 ApplicationResources.properties 中添加内容，在 eclipse 中打开这个文件，添加以下语句（如图 11-52 所示）。

```
login.title=struts_Test02

prompt.userName=<b>Username</b>
prompt.password=<b>Password</b>

button.submit=Submit
button.reset=Reset
```

图 11-52　ApplicationResources.properties 文件

这些内容分别对应了 login.jsp 中五个<bean:message>标签要显示的内容。

现在，将 login.jsp 和 ApplicationResources.properties 两个文件保存。启动 Tomcat，打开浏览器，键入 URL：http://localhost:8080/struts_Test02/login.jsp。

在图 11-53 中看到画圈部分的文字，都是由<bean:message>标签从 Application Resources.properties 中取得并显示的。



图 11-53　运行结果

把 ApplicationResources.properties 的内容补充完整，其余的如下内容是其他 JSP 文件中要用到的（如图 11-54 所示）。

```
login.title=struts_Test02
login.success.title=struts_Test02/success
login.false.title=struts_Test02/false

prompt.userName=<b>Username</b>
prompt.password=<b>Password</b>

button.submit=Submit
button.reset=Reset

login_false.message=Invalid username and/or password, please try again
login_success.message=Welcome
login_false_return.message=Return Login Page
```

图 11-54　ApplicationResources.properties 文件

2. LoginAction.java

源程序如图 11-55 所示。

```
package test.struts.action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import test.struts.database.SelectData;
import test.struts.form.LoginForm;

public class LoginAction extends Action {

  public ActionForward execute(ActionMapping mapping, ActionForm form,
      HttpServletRequest request, HttpServletResponse response)
      throws Exception {

      String userName = ((LoginForm)form).getS_Name();
      String password = ((LoginForm)form).getS_password();

      SelectData o_selectdata = new SelectData();
      boolean b_forward = o_selectdata.selectDataFromDB(userName, password);

      if (b_forward) {
          HttpSession session = request.getSession();
          LoginForm loginform = (LoginForm)form;
          session.setAttribute("login_session",loginform);

          return mapping.findForward("success");
      } else
          return mapping.findForward("false");
  }
```

图 11-55　LoginAction.java 文件

这段程序执行的流程是：

首先，从 ActionForm 中取出用户在 login.jsp 中输入的用户名和密码，它们是由 ActionServlet 取得并存入 ActionForm 中，然后 ActionServlet 再把 ActionForm 作为参数传到 Action 中。只需要调用 ActionForm 类中对应的 getXXX()方法从其中把值取出来就可以了。

然后，把从 ActionForm（这段程序中是 LoginForm）中取得的用户名和密码作为参数调用 SelectData. SelectDataFromDB（String ps_name, String ps_pwd）方法，通过该方法的返回值判断是否是合法的用户。

如果是合法的用户，则把用户名和密码存入 session 中，以便后面的程序使用，并返回“success”通知 Struts 打开对应的 jsp 文件。如果不是合法的用户则返回“false”，登录失败。

3. Login_false.jsp

如果用户没有通过上面的身份验证，那么 Struts 就会打开 login_false.jsp（如图 11-56 所示）文件（因为 LoginAction 返回的是 false）。

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=gb2312">
<title><bean:message key="login.false.title"/></title>
</head>
<body bgcolor="#FFFFFF">
<ul>
  <li>
    <html:link page="/login.jsp">
      <bean:message key="login_false.message"/>
    </html:link>
  </li>
 </ul>
</body>
</html>
```

图 11-56　Login_false.jsp 文件

这个 jsp 文件中只显示一个返回 login.jsp 文件的链接，链接所显示的文字也是通过<bean:message>从 ApplicationResources.properties 中取得的。

4. login_success.jsp

如果用户通过了上面的身份验证，那么 Struts 就会打开 login_success.jsp（如图 11-57 所示）文件（因为 LoginAction 返回的是 true）。

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<html>
<head><meta http-equiv="content-type" content="text/html; charset=gb2312">
<title><bean:message key="login.success.title"/></title>
</head>
<body bgcolor="#FFFFFF">
<bean:message key="login_success.message"/>
<bean:write name="login_session" property="s_Name" scope="session" />
<p>
    <html:link page="/logoff.do">
      <bean:message key="login_false_return.message"/>
    </html:link>
</body>
</html>
```

图 11-57　login_success.jsp 文件

这里的<bean:message>标签就不介绍了。<bean:write name= "login_session" property= "s_Name" scope= "session"/>的作用是从 session 中找名为 "login_session" 的 session，再从 session 中取出名字为 "s_Name" 的属性并输出到 JSP 文件中。"s_Name" 是什么时候被存入 session 的呢？还记得 LoginAction.java 中的如下语句吧？

```
HttpSession session = request.getSession();
LoginForm loginform = (loginform)form;
session.setAttribute("login_session",loginform,);
```

就是在这个时候，从 login.jsp 中取得的用户名和密码被存入到了 session 中。要注意，如果把 Form 存入 session 中，那么<bean:write>中的范围应该是 session，也就是 scope="session"。如果把 Form 存入 request 中，那么范围就应该是 scope="request"了。

这一页中还有一个指向/logoff.do 的链接。

5. LogoffAction.java

源代码如图 11-58 所示。

```
package test.struts.action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
public class LogoffAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception {
            HttpSession session = request.getSession();
            session.removeAttribute("login_session");
            return mapping.findForward("false");
    }
}
```

图 11-58 LogoffAction.java 文件

当用户单击 login_success.jsp 中的指向/logoff.do 的链接的时候，Struts 会根据 struts-config.xml 中的内容自动实例化 LogoffAction 并调用 execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response)方法，在这个方法里面，要实现的功能是将登录时保存在 session 中的内容删除掉。

11.8.9 实例运行

启动 Tomcat，打开浏览器，键入 URL: http://localhost:8080/struts_Test02/login.jsp 在打开的网页中，输入用户名：Tom，密码：struts。单击“提交”，会看到新的页面（见图 11-59）。



图 11-59 实例运行

单击“Return Login Page”，返回 login.jsp。在 login.jsp 中随便输入一个用户名和密码。单击“提交”，会看到登录失败的页面（如图 11-60 所示）。

单击链接同样会回到 login.jsp，但是这两个页面返回 login.jsp 的方法却完全不同。



图 11-60　实例运行

11.8.10　实例小结

现在对本例子中所做的工作总结如下。

（1）三个 JSP 文件，login.jsp，login_success.jsp，login_false.jsp。

（2）一个 ActionForm 类,LogiForm.java。

（3）两个 Action 类，LoginAction.java 和 LogoffAction.java。

（4）一个模拟从数据库中取数据的类，SelectData.java。

它们的工作情形可参阅图 11-61，图中上半部分的虚线框对应的就是 struts-config.xml（用户可以使用任意的文本编辑器打开它）中<action-mappings>标签的这段代码。

```
<action
    attribute="LoginForm"
    name="LoginForm"
    path="/login"
    scope="request"
    type="test.struts.action.LoginAction"
    validate="false">
    <forward name="success" path="/login_success.jsp" />
    <forward name="false" path="/login_false.jsp" />
</action>
```

这一部分所对应的工作过程是，如果客户端浏览器发出 login.do 的请求，那么 Struts 会自动实例化 LoginAction 这个类，并调用其中的 execute()方法，如果 execute()方法返回的是 success，那么 Struts 会打开 login_success.jsp 文件，如果 execute()方法返回的是 false，那么 Struts 会打开 login_false.jsp 文件。

图 11-61 下半部分的虚线框，对应的是 struts-config.xml 中的<action-mappings>标签的<action path="/logoff" type="test.struts.action.LogoffAction" validate="false"/>这段，这一部分所对应的工作过程是，如果客户端浏览器发出 logoff.do 的请求，那么 Struts 会自动实例化 LogoffAction 这个类，并调用其中的 execute()方法。还记得为什么这个标签中没有<forward />吗？是因为有了<global- forwards>标签中的<forward name="false" path="/login.jsp"/>这一段。当 LogoffAction 返回 false，但是在<action path="/logoff"type ="test.struts.action.LogoffAction"validate="false"/>中没有找到对应的<forward/>的时候，<forward name="false" path="/login.jsp"/>就会起作用了，Struts 会根据它来打开 login.jsp 这个文件。

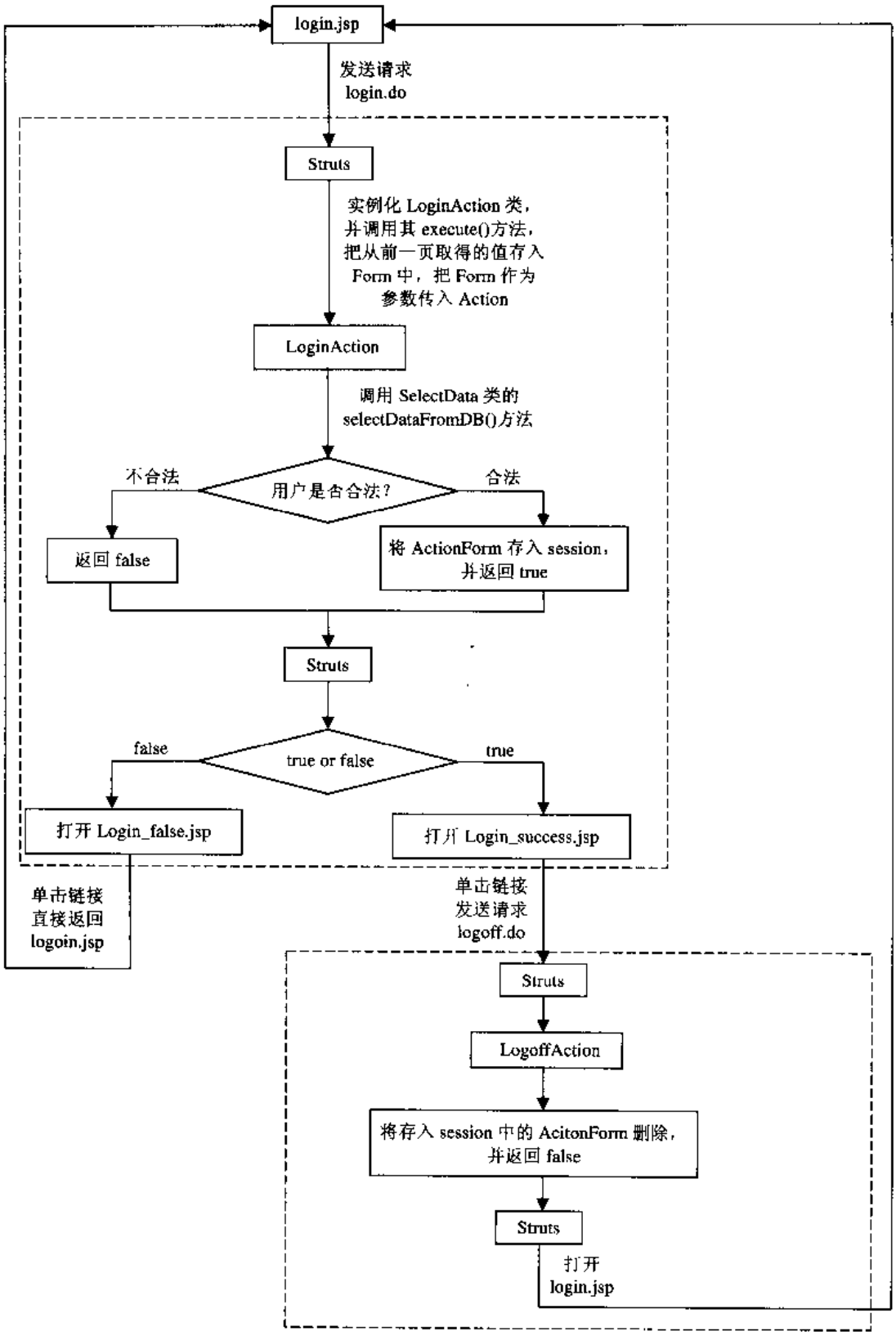


图 11-61　示例程序的工作流程

有关 Struts 的更多的知识，请到 Struts 的官方网站 http://jakarta.apache.org/struts 上寻找。

本章小结

本章通过两个简单的示例程序主要介绍了：

- 什么是 Struts；
- Struts 是怎样工作的；
- 使用 Eclipse 和 Struts 自带的 Web 应用程序来开发简单的程序；
- Struts 的简单工作原理；
- Struts 的标签库；
- 使用专门用于开发 Struts 应用程序的插件 Easy Struts 来开发 Web 应用程序。

第 12 章　将 Eclipse 作为建模工具

作为一种编程工具，Eclipse 有着独有的魅力，吸引着各位使用者。除此之外，Eclipse 还可以作为其他各种领域的工具使用。近年来，实现这些功能的插件陆续登场，引得世人瞩目。其中最受人瞩目的应该是作为 UML 建模工具应运而生的插件“EclipseUML”。本章将介绍该插件的安装，并根据实际设计的流程介绍使用 Eclipse UML 制作 UML 图的方法。在本章里，将会介绍如下内容：

- 何谓 EclipseUML 插件
- EclipseUML 插件的安装
- 利用 EclipseUML 插件绘制 UML 图的基础
- 利用 EclipseUML 插件绘制 UML 图的工程实例

12.1　何谓 EclipseUML 插件

所谓的 EclipseUML（http://www.eclipseuml.com/）插件，就是在 Eclipse 上进行 UML 建模所需要的插件。

一般地，UML 建模工具都具有以下几种功能。

- ML 图的绘图功能
- 根据类图生成源代码的功能
- 根据源代码生成类图的功能
- 生成报告功能

下面，对 EclipseUML 的四项功能进行简单的介绍。

1．UML 图的绘图功能

利用 UML 图的绘图功能，可以绘制下面九种图：

- 用况图（use case diagram）
- 类图（class diagram）
- 顺序图（sequence diagram）
- 协作图（collaboration diagram）
- 组件图（component diagram）
- 部署图（deployment diagram）
- 对象图（object diagram）
- 状态图（statechart diagram）
- 活动图（activity diagram）

作为设计的对象，这些图的重要性自不必说，前面的三种图（用况图、类图、顺序图）在 Web 系统中发挥着重大的作用。在系统的构建中一般包括几个重要的阶段，表 12-1 列举了 EclipseUML 插件在各阶段中的使用情况。

表 12-1　Eclipse UML 插件在系统构建各重要阶段的使用

系统分析阶段	用况图，活动图
概要设计阶段	类图、顺序图、协作图、对象图
详细设计阶段	类图、顺序图、协作图、状态图
编程过程阶段	类图、状态图、组件图、部署图

经过上面的列举，可以看出利用 EclipseUML 插件所能够绘制的图，几乎覆盖了所有使用 UML 进行分析以及设计的全过程。

2．根据类图生成源代码的功能

利用 EclipseUML 插件，在制作类图的同时，可以自动的生成类的 Java 源代码。所生成的只是雏形的雏形，包括 setter 以及 getter 等方法，并且将其显示于“包资源管理器”中。

除此之外，当要对类图的方法名进行变更或者追加属性时，只要进行保存，该变化也会同时反映到源代码之中。相反，如果对源代码进行修改，保存后也会在类图中显示出来。但是，当在方法内进行编码时，如果在类图中将该方法删除的话，那么与其相应的代码也会同时被删除，这一点需要大家注意。一旦错误地进行了删除动作，千万不要将该状态进行保存，在“包资源管理器”中以鼠标右击该方法，然后选择“由本地历史记录恢复”即可将删除的方法复原。

3．根据源代码生成类图的功能

利用 EclipseUML 插件，可以利用输入的源代码生成类图。但是，这样生成的类图，各个项目的位置将会很混乱，需要手动调整。

该功能与“根据类图生成源代码的功能”相关联，但是，与 Java 源代码相关联能够生成的只有类图 -种。在其他的一些商用制品中，还拥有生成其他图的功能。但利用目前的插件还不能够实现根据 Java 源代码生成类图以外其他图的功能。

4．生成报告功能

在生成报告时，可以将所生成的图转换为以下几种格式：

- GIF 格式
- JPEG 格式
- SVG 格式
- WMF 格式

令人遗憾的是目前还不能转换成 XMI 格式，这一点，期待着今后的改善。

12.2 EclipseUML 插件的安装

在本节讲解中使用的 EclipseUML 插件是 2003 年 8 月 6 日发布的版本，该版本可以在 Eclipse 2.1.1 版本中稳定运行，读者可以根据自己的计算机环境进行选择。

下面介绍 EclipseUML 插件的取得、安装以及设定。

可以从 http://www.eclipseuml.com/download.jsp 中得到 EclipseUML 插件。推荐用户将其他必要的插件一并由“Auto-Installer”中下载“eclipseuml-installer_1.2.1.20030806.jar”得到。由于是免费的插件，所以需要事先登录。

下面看怎么进行安装。首先，需要确定计算机上是否安装了 JDK。如果没有安装，请参考第 2 章内容安装 JDK。如果 JDK 已经安装了，单击“开始”→“运行”，在弹出的对话框中添“cmd”→“确认”。然后在弹出的窗口中，变换路径到放置插件的文件夹下，运行：java -jar eclipseuml-installer_1.2.1.20030806.jar 即可启动 GUI 安装程序，如图 12-1 所示，请选择安装语言。



图 12-1 安装启动画面

下面以选择英语为前提向大家进行介绍。

按照安装程序所给的提示，逐步地进行选择。在决定安装路径的时候，选择 Eclipse 的安装目录（本书选择的安装目录为 H:\eclipse）如图 12-2 所示。

选择“Next”出现如图 12-3 的提示框，选择“Yes”即可。

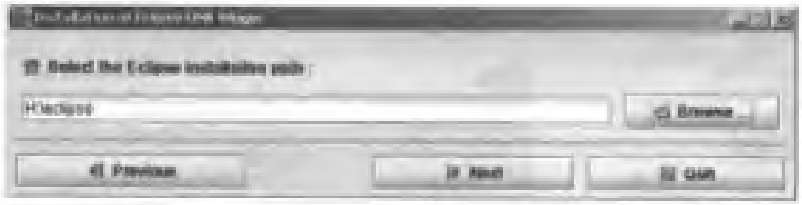


图 12-2　选择 Eclipse 的安装目录

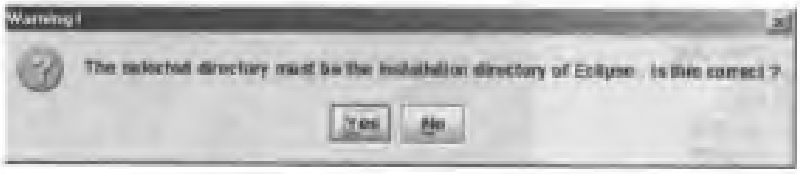


图 12-3　选择“Yes”

安装完毕后，单击显示画面右下方的“Quit”按钮。至此，GUI 安装完毕。

然后，启动 Eclipse，待定的更新管理的设定画面会显示出来，将所有项目选中，单击“完成”即可完成（见图 12-4）。Eclipse 会自动重新启动，此时选择“窗口（W）”→“首选项（P）”，如果显示出 UML 项目栏，则表明安装正常完成（如图 12-5 所示）。



图 12-4　在更新管理中将所有的项目选中

12.3　利用 EclipseUML 插件绘制 UML 图的基础

下面简单介绍如何利用 EclipseUML 插件绘制 UML 图。

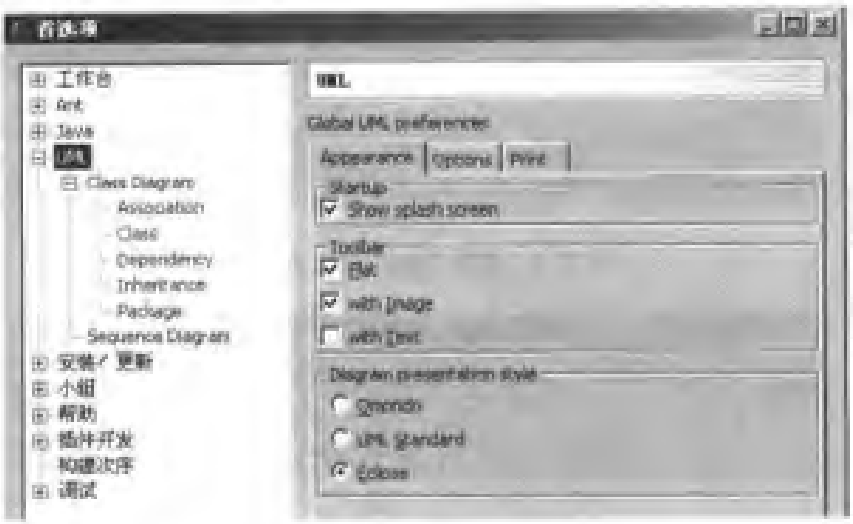


图 12-5　表明安装成功的窗口显示

1. 创建项目以及包的做成

下面举一个简单的 Java 项目的例子，项目名为“PersonSearch”，该项目在下一节也会被使用。具体操作：“文件（F）”→“新建（N）”→“项目（R）”，在弹出窗口的左面选择“Java”，右边选择“Java 项目”→“下一步（N）”→在“项目名”中填“PersonSearch”→“下一步（N）”→“完成（F）”。这样一个 Java 项目就建成了。然后用鼠标右击项目“PersonSearch”→“新建（W）”→“包”，在弹出窗口中填写包名“personData”→“完成（F）”，就可以制成名为“personData”的包了。创建后的情况如图 12-6 所示。



图 12-6　“personData”包

2. 绘制 UML 图的基本操作

（1）新建 UML 图。

用鼠标右击刚刚生成的“personData”包，选择“新建（W）”→“其它（O）”→“UML Diagram”。会显示出九种图的列表，如图 12-7 所示。

（2）选择所要绘制图形的类别。

这里选择“UML Class Diagram”，然后单击“下一步（N）”，在弹出的对话框中填写此图的信息（在 12.4 节中详细讲述了类图的制作方法）。在这里采用系统默认的名称“personData.ucd”，单击“完成（F）”后，一幅新的类图就生成了，如图 12-8 所示。

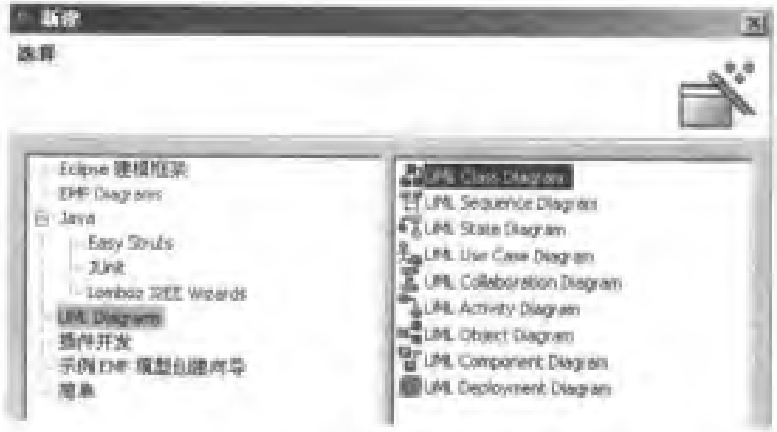


图 12-7　列举了九种可供选择的图

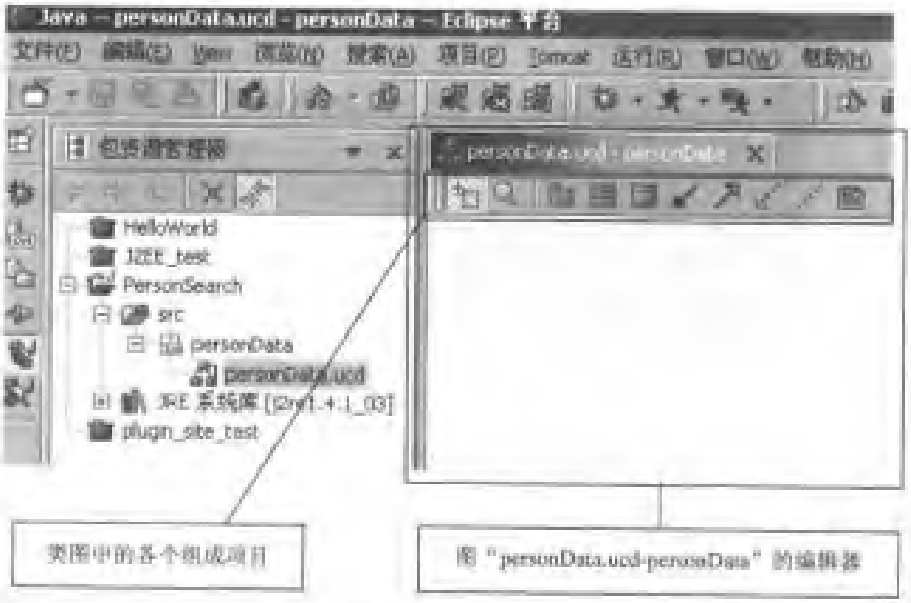


图 12-8　一张新的类图

在图 12-8 中，类图“personData.ucd-personData”的编辑器的上方显示着与所选图形相适应的各种项目以及相关使用的各种线。“Note（说明）”和“Indication（说明同项目的结合）”这两个项目是 UML 图的共同项目，所以在任何一种图中都有所记述。

2. 绘制 UML 图中组成项目的方法

UML 图中组成项目是指组成 UML 图的各个元素，比如说类图是由类、关联等组成。

下面举例说明如何绘制 UML 图的组成项目。比如，要建一个名字叫“Employee”的类，则单击一下“Create a class”（ ），再单击一下编辑器的空位置，弹出一个“新建 Java 类”对话框，在这里输入适当的属性等信息，如图 12-9 所示。

单击“完成（F）”后，一个类就生成了，如图 12-10 所示。当需要连线的时候，先单击选择其中一个项目，然后再单击选择另外一个项目，这时同样会出现一个输入信息的对话框，在对话框中输入相关的信息即可。不能够被关联的项目，或者是利用所选线

无法实现关联的项目，在编辑器中使用不同的颜色显示为无法关联。如果想要对已经存在的项目进行属性信息的输入，双击该项目即可。



图 12-9 “新建 Java 类”对话框



图 12-10 新建成的 Employee 类

3. 位置调整的方法

若需要将一个 UML 图中的一些相关的项目进行上下左右的移动，单击一下该项目，然后按住“Shift”键，再以鼠标右击其他的项目，这时在下拉菜单中会出现一个叫做“Alignment”的选择项。

利用“Alignment”的选择项就可以实现以某个项目为中心，上下左右移动相关项目了。此时是以后点的项目为基准，对先点的项目进行位置的移动。另外，还可以通过单击左上方的“zoom”来实现微调。除此之外，还可以对项目的 Group 化以及前后的重叠进行调整。

12.4　利用 EclipseUML 插件绘制 UML 图的工程实例

在本节中，假定一个“人员信息管理系统”，沿着设计工程的流程进行制图及说明，借此来了解一下使用 EclipseUML 制图的顺序。前面的叙述中曾经说过，在 Web 系统中，经常用到用况图（use case diagram）、类图（class diagram）、顺序图（sequence diagram），因此，在本节中，对这三种图进行比较详细地说明，对其他图仅作简单介绍。

12.4.1　在系统分析阶段中描述用况

在表 12-2 中，列出的是在系统分析阶段中的 UML 图及该图常用的组成项目。

表 12-2　系统分析阶段中的 UML 图及其常用组成项目

UML 图的类型	用况图
可描述的项目	角色（Actor）、用况（Use case）、系统（System）、关联（association）、泛化（generalization）、包含（include）、扩张（extend）

1．制作用况图的方法

用鼠标右键单击包“personData”→“新建（W）”→“其它（O）”→“UML Diagram”→“UML Use Case Diagram”→“下一步（N）”，弹出“New UML Use Case Diagram”对话框。说明如下。

- 输入或选择父文件夹：选择要创建的用况图的位置。这里选择上一节中建好的包“personData”，具体路径是：“PersonSearch/src/personData”。
- 文件名：用况图的名称。这里用系统默认的名称“personData.uud”。
- 包：用况图所在的包。这里用系统默认的包“personData”。

填写完毕后，单击“完成（F）”，则用况图制作完成。

2．创建“角色”（actor）

选择“Create an actor”（），再单击一下图的空位置，弹出“New actor”对话框，在“Name”中填写角色的名字。在这里，填写“部门负责人”，单击“确定”后，生成了一个新的角色。

3．创建“系统”（system）

选择“Create a system”（），再单击一下图的空位置，弹出“New system”对话框，在“Name”中填写系统的名字。在这里，填写“人员信息管理系统”，单击“确定”后，生成了一个新的系统。

4．创建“用况”（Use Case）

选择“Create a use case”（），再单击一下图的空位置，弹出“New use case”对话框，在弹出的对话框中描述用况。在这里，填写“检索”，单击“确定”，至此生成了一个新的用况。将该用况拖到系统“人员信息管理系统”中。此时，用况和系统成为一

组，可以一起进行移动。

5. 创建“关联”（association）

选择“Add an association”（ ），单击一下“检索”，再单击一下“部门负责人”，则“部门负责人”与“检索”之间的关联就建成了。

6. 加入适当的注释

选择“Note”（ ），再单击一下图的空位置，弹出“Note”对话框，在“Input note text”中，填写注释，填写后，单击“确定”，则 Note 制成。可以用“Indication”（ ）将 Note 说明的项目联接起来。

图 12-11 所示为建成后的用况图，从该图中能够了解到系统的使用者，在复杂的用况图中，还可以描述多个用况及各个用况之间的关系。



图 12-11 用 况 图

12.4.2 在概要设计阶段中抽出概念模型

表 12-3 中列出的是在概要设计阶段中的 UML 图，及该图常用的组成项目。

表 12-3 概要设计阶段的 UML 图及其常用组成项目

UML 图的类型	类图
可描述的项目	类（class）、接口（interface）、包（package）、关联（association）、依赖关系（dependency）、泛化关系（generalization）

1. 制作类图的方法

上一节中创建了一个类图 personData.ucd，为使读者有更详细的了解，在这一节中将重新创建类图 personData.ucd。

用鼠标右击包“personData”→“新建（W）”→“其它（O）”→“UML Diagram”→“UML Class Diagram”→“下一步（N）”，此时弹出“New UML Class Diagram”对话框。说明如下。

- 输入或选择父文件夹：选择要创建类图的位置。这里选择建好的包“personData”，

具体路径是“PersonSearch/src/personData”。

- 文件名：类图的名称。这里用系统默认的名称“personData.ucd”。
- Package：类图所在的包。这里用系统默认的包“personData”。
- Select your diagram presentation mode：选择图的表述方式。这里选择“Inheritance”。

填写完毕后，单击“完成”，则类图制作完成。

2. 创建“类”（Class）

上一节中制作了一个名为“Employee”的类，在本节中再来创建一个类“EmployeeList”。选择“Create an class”（ ），再单击一下图的空位置，弹出“新建 Java 类”对话框。说明如下。

- 原文件：类及其所属包的所在位置。这里是“PersonSearch/src”。
- 包：类的所属包。这里是“personData”。
- 封装类型：如果要封装该类则选择该选项，并添加封装类型。
- 名称：类名。
- 修饰符：类的修饰符。
- 超类：类继承的父类。
- 接口：类实现的接口。
- “你想创建那些方法存根？”：选择想要生成的方法。

如图 12-12 所示，填写好以上信息后，如果单击“完成（F）”，则类的创建就会完成。如果单击“下一步（N）”，则可以选择“Stereotype”，如图 12-13 所示。在这里直接选择“完成（F）”，则类“EmployeeList”就建好了。

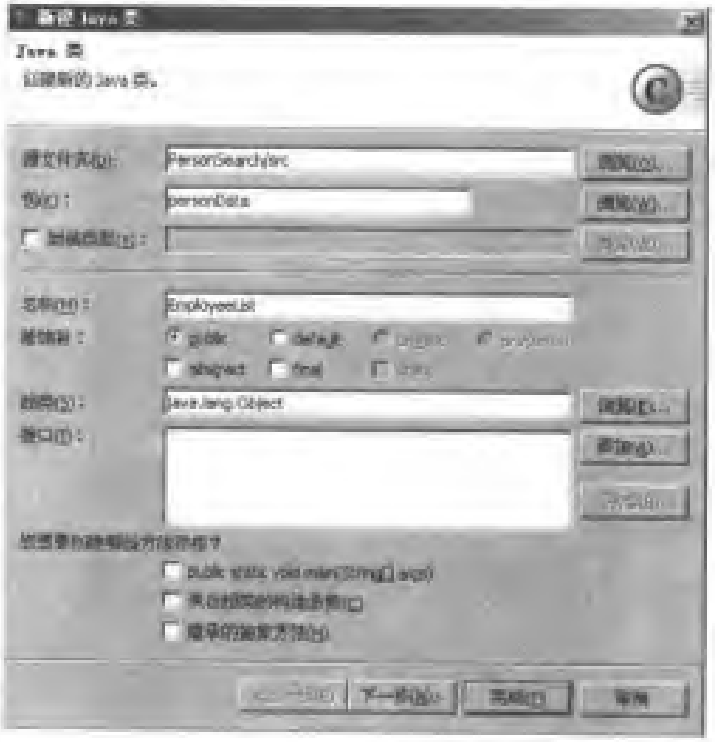


图 12-12　新建 Java 类



图 12-13　选择 Stereotype

3. 创建“关联”（Association）

选择“Association”（✔），单击一下“Employee”，再单击一下“EmployeeList”，弹出对话框“Properties”。在该对话框中，选项“Connection”为两个类关联的信息。“1st Association End”表示类“Employee”的信息，“2nd Association End”表示类“EmployeeList”的信息。下面以填写“1st Association End”为例。

- Navigable：导航。
- Name：在该关联中，类“Employee”的角色名填“employee”。
- Type：该关联端的属性。
- Multiplicity：该关联端的多重性。
- Association type：该关联端的类型。分 Aggregation（聚合）和 Composition（组合）两种。

如图 12-14 所示，单击“确认”后关联就建立好了。

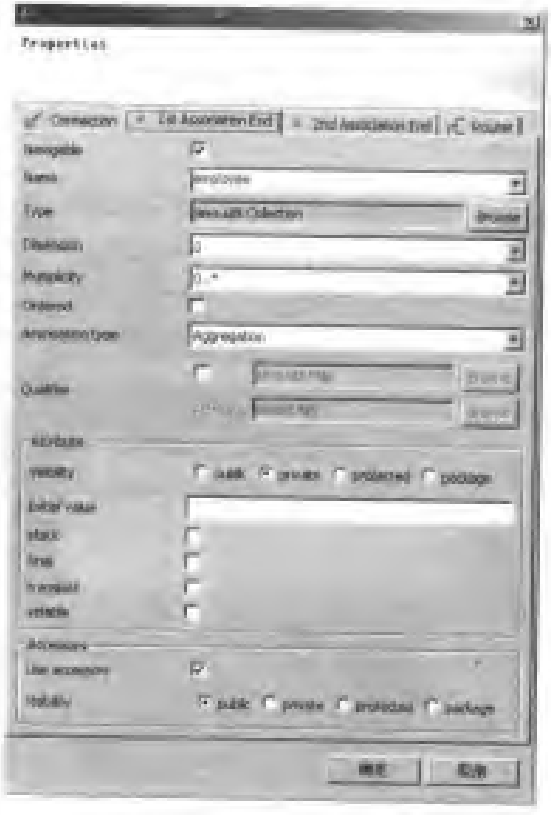


图 12-14　“1st Association End”

按照“关联”的建立方法，建立了为利用 UML 进行设计的基础图，如图 12-15 所示。随着对该图的随时修改，会做出更加详细的类图。

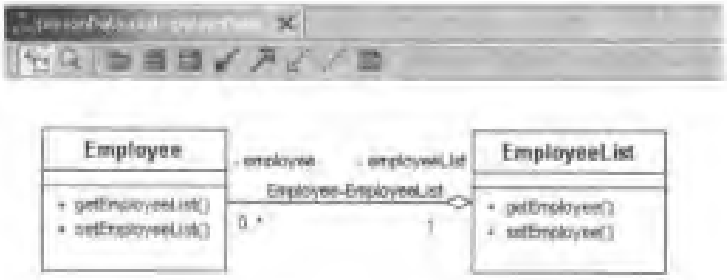


图 12-15　类图

其他的相关操作方法，诸如 Package 和接口等基本相同。

想要变更为 UML 通常使用的+、-、#时，可以将“窗口（W）”→“首选项（P）”→“UML”→“Appearance”路径下的“Diagram presentation style”，由默认的“Eclipse”变更为“UML Standard”。类的自身的显示方式可以通过鼠标右键进行选择。首先，可以选择是“Maximize”还是“Minimize”，如果选择了“View Selector”的话，还可以选择是否显示属性或者方法等。能够视用途来进行变化，这是一个突出的优点。

保存该类图，进行下面的操作。

12.4.3　在概要设计阶段中根据概念模型绘制顺序图

在表 12-4 中，列出的是在概要设计阶段中的 UML 图，及该图常用的组成项目。

表 12-4　概要设计阶段是的 UML 图及其常用组成项目

UML 图的类型	顺序图
可描述的项目	角色（Actor）、对象（Object）、消息（Message）、自调用（self call）、回送（return）

1. 制作顺序图的方法

用鼠标右击包“personData”→“新建（W）”→“其它（O）”→“UML Diagram”→“UML Sequence Diagram”→“下一步（N）”，弹出“New UML Sequence Diagram”对话框。说明如下。

- 输入或选择父文件夹：选择要创建的顺序图的位置。这里选择已建好的包“personData”，具体路径是“PersonSearch/src/personData”。
- 文件名：顺序图的名称。这里用系统默认的名称“personData.usd”。
- Package：顺序图所在的包。这里用系统默认的包“personData”。

填写完毕后，单击“完成（F）”，则顺序图制作完成。

2. 向顺序图添加项目

以刚刚作出的概念模型的类为基础，描述处理的流程。考虑到实际的具体检索处理

的流程，按照以下的顺序作图。

（1）利用“Create an actor”（）做成“操作者”。

（2）利用“Add a new object”（）做成“检索管理画面”和“检索管理”（type:Object）。

（3）利用“Add a new object”在概念模型中生成的两个类，并分别制定类型，然后作成 Object。

（4）利用“Add a message”（）分别追加各自 object 之间的 message。

在输入 object 生成时的属性时，必须事先指定类的类型（见图 12-16）。此时，输入框中没有显示任何内容，但是如果输入了一个字母，那么以该字母为开头的可选项的类就被显示出来。每一次输入字母的时候，都会根据输入的字母进行重新选择。



图 12-16　指定类的类型

追加 message 的时候，首先需要对“label”进行描述。需要将已经做成的类指定为传达目的的时候，由于在 Operation 栏中已经排列出做成的方法，所以不必重写，仅需选择即可。

制成后的顺序图如图 12-17 所示。

也可以单独做成一个类的状态图。但是当单独做成时，适当地变更一下所作成的顺序图的名字，那么就可以与该操作合作使用了。方法与上面的相同，以鼠标右击所选类，然后选择“Create/Open state diagram”，同时也可以制作出与该类相关的状态图。

12.4.4　在详细设计阶段中根据所制成的顺序图补充类图

这一步骤地记述将比先前记述的概念类图更加详细。

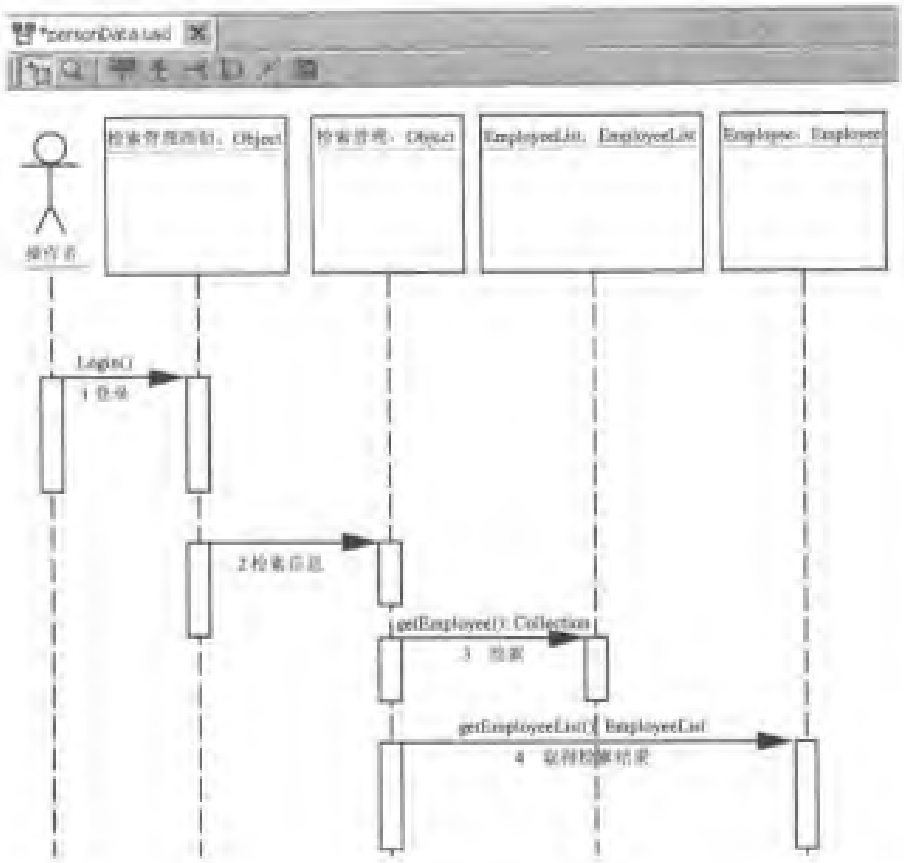


图 12-17　顺序图

下面根据 12.4.3 节的顺序图来补充 12.4.2 节的类图，步骤如下：

（1）追加新类，编辑属性；

（2）追加方法，属性；

（3）关联。

绘制类图的方法与 12.4.3 节中的概要设计阶段基本相同，在这里将其细化为以下几点。

（1）如果要描述“stereotype”，则需要编辑已有类的属性，或者新建类时进行添加，如图 12-18 所示。“SearchControl”的 stereotype 为“control”，在图上表示为“<<control>>”。

（2）制成方法时，以鼠标右击该类，选择“New”→“Method”。这时，会有三个标签显示出来，在“Properties”标签中做好各种指定的工作。还可以在“Implementation”标签中描述方法的实体部分，在“Javadoc”标签中描述 Javadoc 所使用的文件，记述完后，这些相关的信息将如实地反映在源代码之中。

例如，为“EmployeeList”追加“getEmployee”方法，用鼠标右键单击类“EmployeeList”，在弹出的下拉框中选择“New”→“Method”。然后，在弹出的窗口中编辑方法，下面分别介绍各项内容。

- Name：方法名。
- Return type：返回值类型。

- Multiplicity：多重性。
- Visibility：可见性。其中：public 为公有的，在 UML 中用符号“+”表示；protected 为受保护的，在 UML 中用符号“#”表示；package 为 Java 特有的，表示只有该类所在包中的成员可以访问该方法，在 eclipseUML 中用“~”表示；private 为私有的，在 UML 中用“-”表示，值得注意的是这个表示符号与 package 的表示符号一样，它们之间的区别只有在可成的代码中才可以表现出来。
- Parameters：添加方法的参数。
- Exceptions：添加异常。
- Modifiers：方法的修饰符。

（3）制成属性时，以鼠标右击该类，选择“New”→“Attribute”。同生成方法时的步骤相同，需要在“Properties”标签中作各种指定。如果有必要的话，请同时在“Javadoc”标签中描述必要的信息。Inner 类和 Inner 接口也按同样的方法制成。

（4）在变更方法名或者属性名时，直接将其指定，然后在“Properties”标签中进行修改。更改类名的时候，则需要从“Refactor”中选择“Rename Elements”。但是，一旦所制成的属性名被更改，与原来的属性名相对应的在源代码中生成的 getter，setter 方法将仍然留在源代码中，所以这部分的工作需要用户自行删除。

（5）在绘制关联关系的线时，需要在“Router”标签中，首先在“Manual”和“Fixed at edge”中决定类和线的接点，然后，通过变更“Manhattan”来进行调整，这样绘制出的图会显得更漂亮。

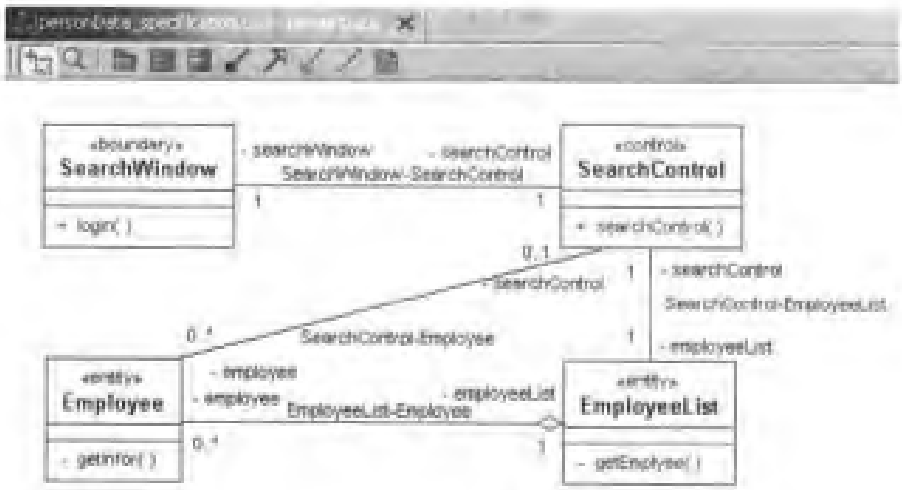


图 12-18 类图

12.4.5 在“编程过程”中整理 Package

绘制 package 图的过程其实也就是绘制类图的过程，只不过是需要决定如何对已有的某个类进行处理。这里使用已经作成的所有类（共六个）。在刚才的类图中，需要进行如下新的操作。

（1）利用“Create a package”生成“personData.view”，“personData.control”，“personData.employee”这几个 package。

（2）以鼠标右击所选类，利用“View Selector”使方法隐藏，并分别移动到 package 内。

（3）在 package explorer 之间进行文件移动。

（4）package 之间的关联连接。

执行步骤（2）后，会出现如图 12-19 所示的画面。如果至此就进行保存的话，那么所绘制的图仅能反映一些类的信息。接下来，还需要在“包资源管理器”中，将已有的“...java”拖入各自的 package 内。这样，有关构成的信息就都可以显示在类图和源代码中了。

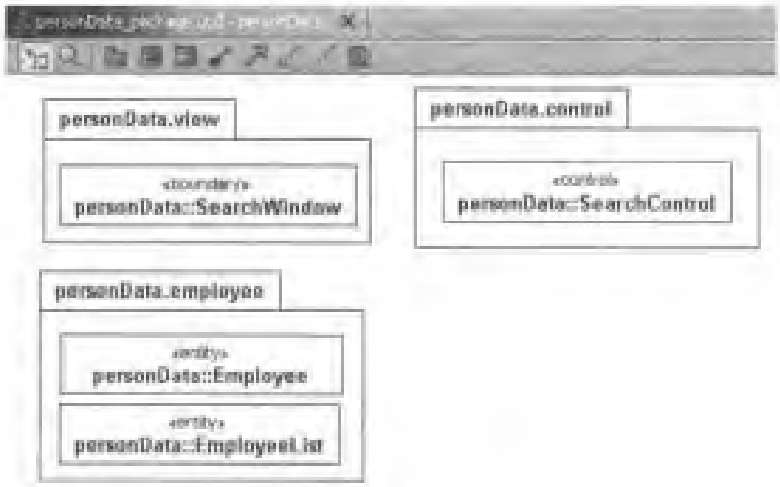


图 12-19　包的情况

执行步骤（4）的结果如图 12-20 所示，该图表明了 Package 之间的关联连接。

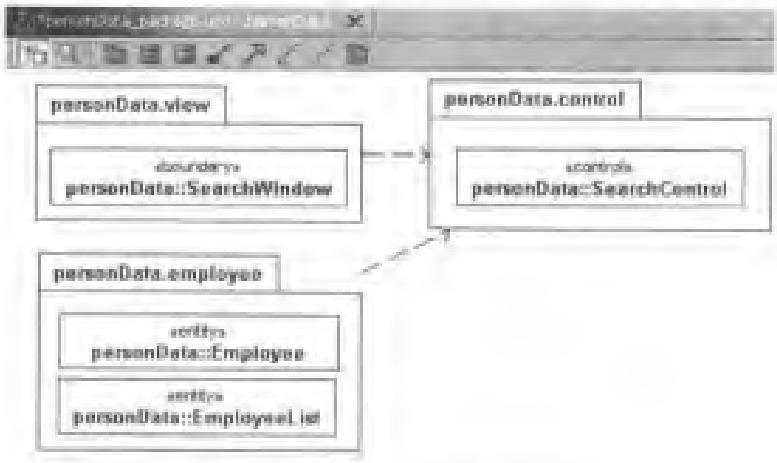


图 12-20　Package 之间的关联连接

在本次的流程之中，是一边制作类，一边抽象归纳出包并对其进行分类。如果从决

定所要构成的组件开始进行详细的类设计的话，也会出现先绘制 package 图的情况。这种情况下，可以在类图中，以鼠标右击编辑器，通过“Insert”→“Package”来实行。

12.4.6 在“编程过程”中分割组件，整理部署

在表 12-5 中，列出的是在“编程过程”中的 UML 图，及该图常用的组成项目。

表 12-5 “编程过程”中的 UML 图及其常用组成项目

UML 图的类型	组件图、部署图
可描述的项目	组件（component）、接口（interface）、泛化（generalization）、关联（association）、依存（dependency）

组件图（component diagram）：表明系统中各组件及其关系，如图 12-21 所示。

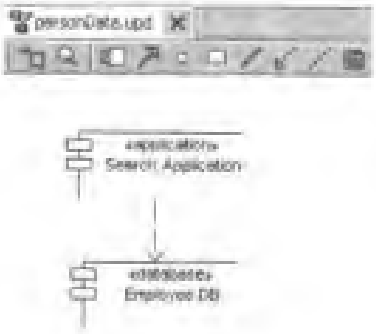


图 12-21 组件图

部署图（deployment diagram）：表明系统中软件组件和硬件组件之间的物理关系。如图 12-22 所示，此系统有两台服务器：应用服务器和数据库服务器。其中，应用服务器用来存放应用程序，数据库服务器用来存放应用程序所需的数据。

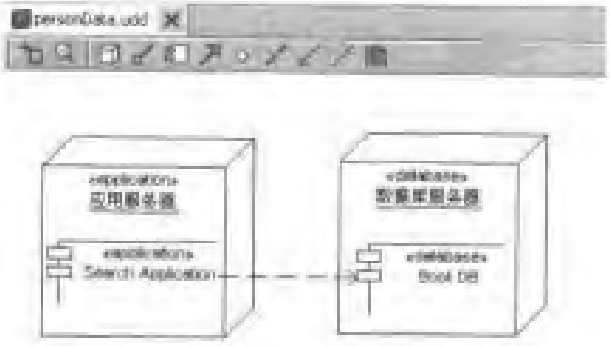


图 12-22 部署图

以上仅是一个沿着工程使用进度而对所要使用图的粗略介绍。下面对其他图进行简要介绍。

12.4.7　其他图简介

1. 协作图（collaboration diagram）

协作图（见图 12-23）是两种交互图的一种。交互图分为顺序图和协作图。其中表示对象之间交互行为的时间顺序的交互图是顺序图，而表示对象之间交互行为关系的交互图是协作图。协作图与顺序图从不同的角度反映相同的信息。



图 12-23　协作图

协作图中的元素有：

- 角色（actor）
- 对象（object）
- 消息（message）

消息的完整语法如下：

```
    perdecessoropt guard-conditionopt sequence-expressionopt return-value-list:
= opt message-name(argument[,at])
```

其中：perdecessor——前驱，表明该消息的前驱；guard-condition——条件表达式；return-value-list——返回值列表；message-name——消息名；argument——参数列表。

可以制作两种协作图，一种是式样等级的组件图，另一种是加入角色的接口登记的组件图。

2. 对象图（object diagram）

对象图也称为实例图，它反映了系统某一时间对象情况。

对象图是由协作图生成的没有信息的图。协作图展示的是动态的方面，与此相反，Object 图展示的则是静态的方面。

3. 状态图（statechart diagram）

状态图，是一种用来表述系统行为的 UML 图。通常，表示单个对象在其整个的生

存周期内的行为过程和对象各个时期的状态。

状态图中的元素有：

- 状态（state）——对象在不同时期的情况一般分为简单状态和组合状态。
- 转接（transition）——事件[条件] / 动作（英文表示为 Event [Guard] / Action）。转接中有一种是自转接。

如图 12-24 所示，用状态图来显示数据处理的内部状态，这个简单的例子可以涵盖状态图的大部分的语法。

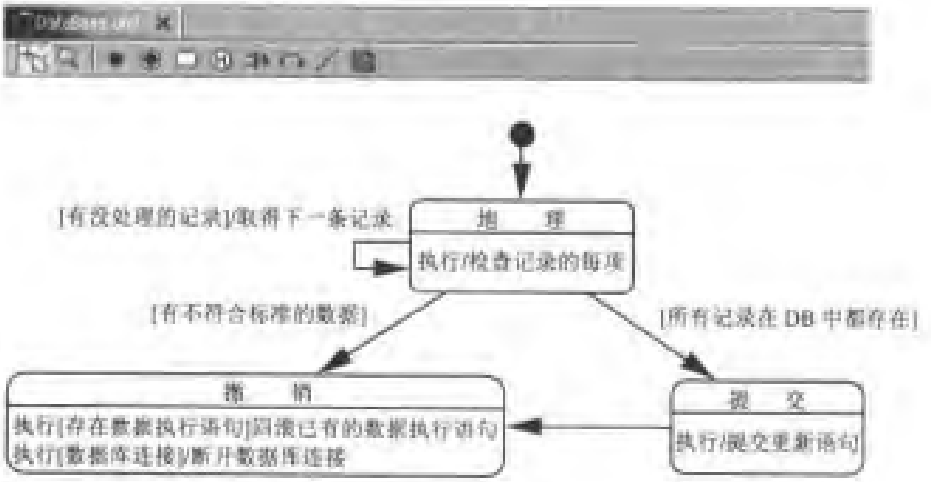


图 12-24　状态图

4. 活动图（activity diagram）

活动图类似于 State Chart 图，活动图能够确定工作流程，如图 12-25。

活动图中的元素有：

- 活动状态（activity state），简称活动（activity），它反映了动作的状态
- 分支（branch）与合并（merge）
- 分岔（fork）与汇合（join）
- 转接（transition）
- 信号（signal）
- 区域（partition）

虽然活动图在 UML 的九种图中最不被提倡用，但是在制作 Web 应用程序时，却经常会使用到活动图。由于活动图的项目在 Toolbar 中同 State Chart 图的项目重合，因此在使用的时候请注意区分。可以将每一项活动做成单独的一个路径，并在其中进行描述。

本章小结

在本章中讲解了 EclipseUML 插件及其安装，以及利用 EclipseUML 插件绘制 UML 图的有关知识。

自统一建模语言 UML 问世以来，虽然各方面的评价褒贬不一，但许多成功的实践说明 UML 的确是一种优秀的建模语言。读者想要真正体会到 UML 的精妙，需要仔细体会面向对象建模的思想，并要长期地努力，不断地学习、实践。

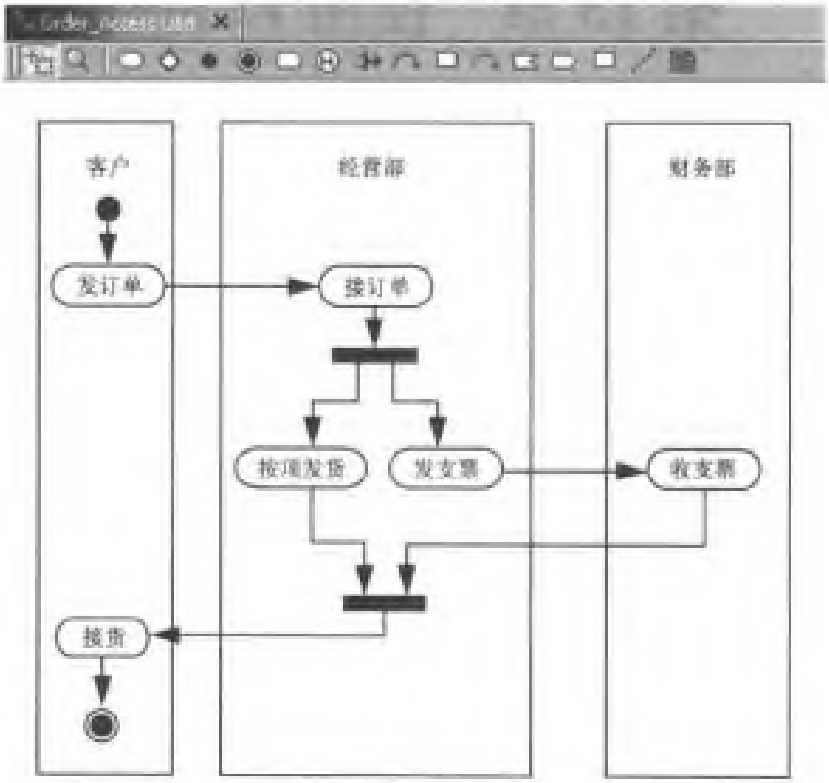


图 12-25　活动图

第 13 章　使用 Eclipse 进行重构

本章中所涉及的重构工作需要读者在实际的编程工作中仔细地体会。本章将介绍如下内容：

- 什么是重构
- 为什么要进行重构
- 使用 Eclipse 进行重构的实例

13.1　什么是重构

这里所说的重构（Refactoring），就是指“保持外部状态和功能不变，而优化内部处理编码”的方法。

说到重构，就不得不提一下 Martin Fowler 所著的《Refactoring:Improving the Design of Existing Code》一书（中译本《重构——改善既有代码的设计》，侯捷、熊节译，中国电力出版社）。Eclipse 重构工具就支持该书中描述的许多变换，如抽取方法和内联局部变量等。

13.2　为什么要进行重构

每一个系统都有自身的生命周期。这个“生命”诞生于编码伊始，成长于版本更替。大多数的人都希望第一个版本就能够正确而全面地满足人们需要，但实际上，这几乎是不可能的。每一个系统都会经历功能扩充。为了适应将来的功能扩充，在进行开发的过程中往往会采取以下两种方式。

（1）充分考虑到将来出现的功能扩充，进行深入设计。

（2）为了适应将来可能发生的功能扩充，而将现在的代码编得尽量简单易懂。

极限编程主要采取第二种方式，为了达到这种技术的要求，这种程序开发的方法更能够适应未来功能需求的变化。而无论是第一种方式还是第二种方式都会或多或少地使用到重构。

采用 Eclipse 提供的重构工具能够快速、安全地完成重构任务。以编者本人的体会来说，在进行重构时适当使用 CVS 进行版本控制是必要的，这样做不仅能够恢复过去的版本，而且会使重构工作进行得有条不紊，具体的好处还需要读者在实践中仔细体会。

13.3　使用 Eclipse 进行重构的实例

现举一个实例来说明利用 Eclipse 的重构工具来进行重构实在不是一件容易的事情，实际项目的处理流程太复杂，如果只拿出一部分来讲解，读者理解起来会很费劲。因此这里举一个书友会付费程序的简单实例。加入书友会的会员通过邮寄的方式购买图书会获得一定的优惠。本例主要是计算会员邮购图书的费用。由系统告诉程序：会员购买了哪些书，程序再根据书和会员的信息来计算会员要付的费用和会员的积分。其中，会员分为“黄金会员、白金会员、一般会员”三种。

下面列出从该需求中抽象出来的三个类：购买（Purchase）、书（Book）、会员（Member）。

1. 购买类（Purchase）

```
public class Purchase {
  private Book book;  //书
  Purchase(Book book){
    this.book = book;
  }

  public Book getBook() {
    return book;
  }
}
```

2. 书类（Book）

```
public class Book {
  private String bookName = "";    //书名
  private float bookPrice = 0;     //书价
  private float discount = 0.0f;   //折扣

  public Book(String pBookName, float pBookPrice, float pDiscount) {
    bookName = pBookName;
    bookPrice = pBookPrice;
    discount = pDiscount;
  }

  public String getBookName() {
    return bookName;
  }

  public float getBookPrice() {
    return bookPrice;
  }

  public float getDiscount() {
    return discount;
  }
}
```

3. 会员类（Member）

```
public class Member {
  private String _memberName = "";//会员名
```

```
    private int _memberType = 3;  //会员类型
    private int _totalMark = 0;  //积分
    private ArrayList _purchase = new ArrayList();//购买

    public Member(String memberName) {
      this._memberName = memberName;
    }

    public void addPurchase(Purchase purchase) {
      _purchase.add(purchase);
    }

    public double calculateBooksPrice() {
      double priceAmount = 0;  //购买总金额
      Purchase purchase = null;  //当前购买
      ListIterator iter = this._purchase.listIterator();

      //设定积分
      if(this._purchase.size() >= 5) {
        this._totalMark += this._purchase.size() + 2;
      } else {
        this._totalMark += this._purchase.size();
      }

      //设定会员类型 黄金会员：1；白金会员：2；一般会员：3
      if( this._totalMark>=100 ) {
        this._memberType = 1;
      } else if( this._totalMark>=50 && this._totalMark<100 ) {
        this._memberType = 2;
      } else {
        this._memberType = 3;
      }

      //循环取出本次购买的书籍
      while(iter.hasNext()) {
        purchase = (Purchase)iter.next();
        switch(this._memberType) {
        case 1:
          //黄金会员
          priceAmount += (float)purchase.getBook().getBookPrice() *
      purchase.getBook().getDiscount() * 0.8f;
          break;
        case 2:
          //白金会员
          priceAmount += (float)purchase.getBook().getBookPrice() *
      purchase.getBook().getDiscount() * 0.9f;
          break;
        case 3:
          //一般会员
          priceAmount += (float)purchase.getBook().getBookPrice() *
      purchase.getBook().getDiscount();
          break;
        }
      }

      return priceAmount;
    }
}
```

13.3.1　抽取方法

读者可能已经看出这三个类中存在的一些问题。下面通过 Eclipse 的重构工具，结合“重构”的思想，来重新构造它们。

在这三个类中最容易引起人们注意的就是第三个类 Member 了，因为它有一个很长的方法：calculateBookPrice。每一位看了该方法的开发人员都会敏感地意识到在该方法中实现的功能太多了：设定读者类型、设定会员类型、计算书价。这样做会使程序的可扩展性和易读性变得很差。改善这种状况的方法是将这些功能抽出来作为单独的方法，可以使用 Eclipse 重构工具中的“抽取方法（E）...”很容易地完成这项工作。

首先，看 calculateBookPrice 方法中的如下代码：

```
//设定积分
if(this.purchase.size() >= 5) {
  totalMark += this.purchase.size() * 2;
} else {
  totalMark += this.purchase.size();
}
```

该段代码实现积分的设定，可以将这段程序作为单独的方法，在需要设定积分时只需要调用该方法就可以了。方法是：选中这段程序→右击鼠标→“重构（T）”→“抽取方法（E）...”，如图 13-1 所示。



图 13-1　抽取方法

然后，弹出一个“抽取方法”的对话框，如图 13-2 所示。在该对话框中可以输入方法名，选择访问修饰符，以及是否将抛出的运行时异常添加至方法特征符。

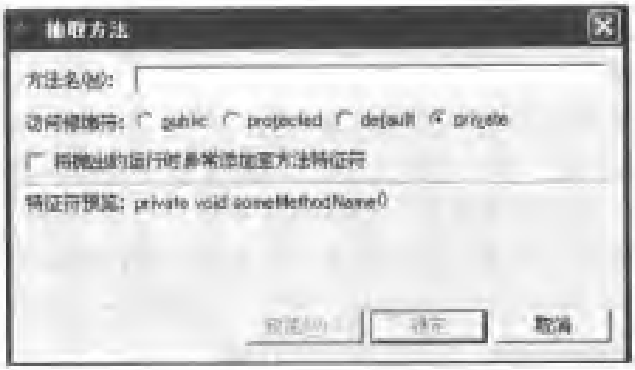


图 13-2 “抽取方法”的对话框

给方法起名为：setTotalMark，如果选择“预览〈W〉>”按钮，则会出现一个窗口来显示重构前和重构后程序的差异，如图 13-3 所示。



图 13-3 重构前和重构后程序的差异

单击“确定”按钮后，该重构就完成了。

按照这个方法再来抽出“设定会员类型”方法，重构后的程序如下：

```
public class Member {
  private String _memberName = "";  //会员名
```

```
    private int _memberType = 3;  //会员类型
    private int _totalMark = 0;  //积分
    private ArrayList _purchase = new ArrayList();//购买

    public Member(String memberName) {
      this._memberName = memberName;
    }

    public void addPurchase(Purchase purchase) {
      _purchase.add(purchase);
    }

    public double calculateBooksPrice() {
      double priceAmount = 0;  //购买总金额
      Purchase purchase = null;  //当前购买
      ListIterator iter = this._purchase.listIterator();

      setTotalMark();

      setReaderYype();

      //循环取出本次购买的书籍

      while(iter.hasNext()) {
        purchase = (Purchase)iter.next();
        switch(this._memberType) {
          case 1:
            //黄金会员
            priceAmount += (float)purchase.getBook().getBookPrice() *
                           purchase.getBook().getDiscount() * 0.8f;
          break;
        case 2:
          //白金会员
          priceAmount += (float)purchase.getBook().getBookPrice() *
                         purchase.getBook().getDiscount() * 0.9f;
          break;
        case 3:
          //一般会员
          priceAmount += (float)purchase.getBook().getBookPrice() *
                         purchase.getBook().getDiscount();
          break;
        }
      }

      return priceAmount;
    }
  private void setReaderYype() {
      //设定会员类型:黄金会员为 1,白金会员为 2,一般会员为 3
      if( this._totalMark>=100 ) {
        this._memberType = 1;
      } else if( this._totalMark>=50 && this._totalMark<100 ) {
        this._memberType = 2;
      } else {
        this._memberType = 3;
      }
    }

    private void setTotalMark() {
      //设定积分
      if(this._purchase.size() >= 5) {
        this._totalMark += this._purchase.size() + 2;
```

```
        } else {
          this._totalMark += this._purchase.size();
        }
      }
    }
```

13.3.2 抽取常量

会员类型有：黄金会员、白金会员、一般会员。程序中使用的是数字 1，2，3，程序中有两处调用它们，在真实的项目中也许会有多个类有这样的调用。这样做是不好的，因为程序的可读性和可修改性差。每当出现这样的问题，应将这些数字或文字用常量替换掉。

使用 Eclipse 重构工具中的“抽取常量（A）...”，可以很容易地将这些数字或文字转化成常量。方法是：选中要抽取常量的数字或文字→右击鼠标→“重构（T）”→“抽取常量（A）...”。对“黄金会员（1）”进行重构，如图 13-4 所示。



图 13-4 对“黄金会员（1）”进行重构

然后，弹出一个“抽取常量”的对话框，如图 13-5 所示。在该对话框中可以输入常量名，选择访问修饰符（这里选择 public，这是因为其他的类将用到这个常量）及两种引用方式：

- 将所有出现了所选表达式的地方都替换为对常量的引用（R）；
- 使用类名限定常量引用（Q）。

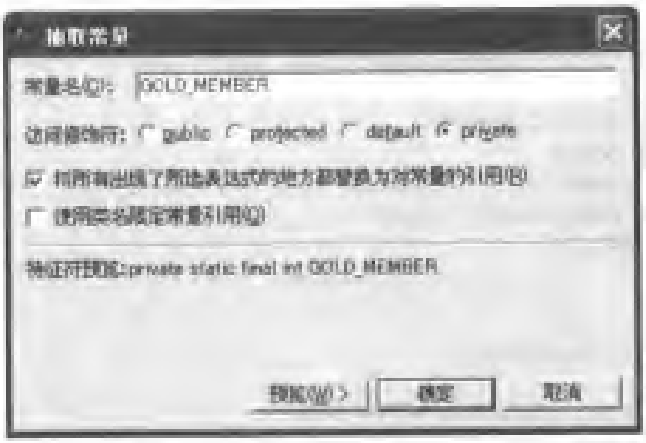


图 13-5　“抽取常量”的对话框

单击确认后，再对其他两个类型（白金会员、一般会员）进行重构，并添加注释。重构后的程序如下：

```
public class Member {

  public static final int GOLD_MEMBER = 1; //黄金会员
  public static final int PLATINUM_MEMBER = 2; //白金会员
  public static final int GENERAL_MEMBER = 3;  //一般会员

  private String _memberName = "";//会员名
  private int _memberType = GENERAL_MEMBER;  //会员类型
  private int _totalMark = 0;  //积分
  private ArrayList _purchase = new ArrayList();//购买

  public Member(String memberName) {
    this._memberName = memberName;
  }

  public void addPurchase(Purchase purchase) {
    _purchase.add(purchase);
  }

  public double calculateBooksPrice() {
    double priceAmount = 0;  //购买总金额
    Purchase purchase = null;  //当前购买
    ListIterator iter = this._purchase.listIterator();

    setTotalMark();

    setReaderType();

    //循环取出本次购买的书籍
    while(iter.hasNext()) {
      purchase = (Purchase)iter.next();
      switch(this._memberType) {
        case GOLD_MEMBER:
          //黄金会员
          priceAmount += (float)purchase.getBook().getBookPrice() *
purchase.getBook().getDiscount() * 0.8f;
          break;
        case PLATINUM_MEMBER:
```

```
            //白金会员
            priceAmount += (float)purchase.getBook().getBookPrice() *
purchase.getBook().getDiscount() * 0.9f;
            break;
          case GENERAL_MEMBER:
            // 一般会员
            priceAmount += (float)purchase.getBook().getBookPrice() *
purchase.getBook().getDiscount();
            break;
          }
      }

      return priceAmount;
    }

    private void setReaderYype() {
      //设定会员类型：黄金会员为1，白金会员为2，一般会员为3
      if( this._totalMark>=100 ) {
        this._memberType = GOLD_MEMBER;
      } else if( this._totalMark>=50 && this._totalMark<100 ) {
        this._memberType = PLATINUM_MEMBER;
      } else {
        this._memberType = GENERAL_MEMBER;
      }
    }

    private void setTotalMark() {
      //设定积分
      if(this._purchase.size() >= 5) {
        this._totalMark += this._purchase.size() + PLATINUM_MEMBER;
      } else {
        this._totalMark += this._purchase.size();
      }
    }
}
```

13.3.3 移动

在现在的程序中价格是按照会员的类型进行计算的：黄金会员在折扣的基础上再打8折，白金会员在折扣的基础上再打9折，一般会员不再有折扣。但是，如果用户的需求发生变化了，比如计算书价的公式发生变化，那么在现在的程序中修改，会比较困难。下面对程序继续进行重构。

首先，使用重构工具的“抽样方法”将下面的语句生成“getCharge”的方法。

```
    priceAmount+=(float)purchase.getBook().getBookPrice()*purchase.getBook().g
etDiscount();
```

然后，使用“查找/替换”，将程序中上面的内容替换成方法 getCharge。修改后程序如下：

```
public class Member {
  ......
  public double calculateBooksPrice() {
    double priceAmount = 0;  //购买总金额
    Purchase purchase = null;  //当前购买
```

```
        ListIterator iter = this._purchase.listIterator();

        setTotalMark();

        setReaderType();

        //循环取出本次购买的书籍
        while(iter.hasNext()) {
          purchase = (Purchase)iter.next();
          switch(this._memberType) {
            case GOLD_MEMBER:
              // 黄金会员
              priceAmount = getCharge(priceAmount, purchase) * 0.8f;
              break;
            case PLATINUM_MEMBER:
              //白金会员
              priceAmount = getCharge(priceAmount, purchase) * 0.9f;
              break;
            case GENERAL_MEMBER:
              //一般会员
              priceAmount = getCharge(priceAmount, purchase);
              break;
          }
        }

        return priceAmount;
      }

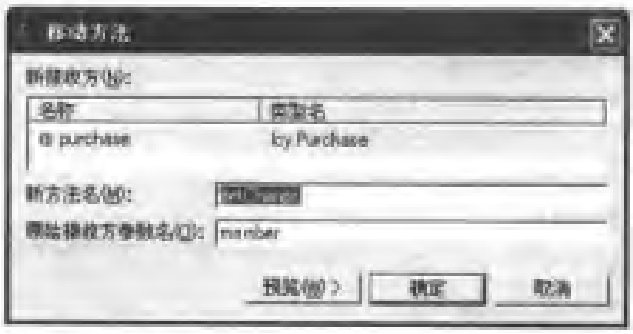
      private double getCharge(double priceAmount, Purchase purchase) {
        priceAmount += (float)purchase.getBook().getBookPrice() *
                       purchase.getBook().getDiscount();
        return priceAmount;
      }
    }
```

发现方法 getCharge 中如下的部分并不应该属于类 Member，应该属于类 Purchase。下面用 Eclipse 重构工具中的“移动（V）...”，将方法 getCharge 移动到类 Purchase 中。方法是：选中“getCharge”→右击鼠标→“重构（T）”→“移动（V）...”，如图 13-6 所示。



图 13-6　移动

然后，弹出一个“移动方法”的对话框，如图 13-7 所示。在该对话框中可以输入新方法名和原始接收方参数名。



如图 13-7 “移动方法”的对话框

单击确定后，程序如下：

```
public class Member {

  public static final int GOLD_MEMBER = 1;        //黄金会员
  public static final int PLATINUM_MEMBER = 2;    //白金会员
  public static final int GENERAL_MEMBER = 3;     //一般会员

  private String _memberName = "";//会员名
  private int _memberType = GENERAL_MEMBER;  //会员类型
  private int _totalMark = 0;  //积分
  private ArrayList _purchase = new ArrayList();  //购买

  public Member(String memberName) {
    this._memberName = memberName;
  }

  public void addPurchase(Purchase purchase) {
    _purchase.add(purchase);
  }

  public double calculateBooksPrice() {
    double priceAmount = 0;  //购买总金额
    Purchase purchase = null;  //当前购买
    ListIterator iter = this._purchase.listIterator();

    setTotalMark();

    setReaderType();

    //循环取出本次购买的书籍
    while(iter.hasNext()) {
      purchase = (Purchase)iter.next();
      switch(this._memberType) {
        case GOLD_MEMBER:
          // 黄金会员
          priceAmount = getCharge(priceAmount, purchase) * 0.8f;
          break;
        case PLATINUM_MEMBER:
          //白金会员
          priceAmount = getCharge(priceAmount, purchase) * 0.9f;
          break;
        case GENERAL_MEMBER:
```

```
            // 一般会员
            priceAmount = getCharge(priceAmount, purchase);
            break;
          }
        }

        return priceAmount;
      }

      private double getCharge(double priceAmount, Purchase purchase) {
          return purchase.getCharge(priceAmount);
      }

      private void setReaderYype() {
        //设定会员类型：黄金会员为 1，白金会员为 2，一般会员为 3
        if( this._totalMark>=100 ) {
          this._memberType = GOLD_MEMBER;
        } else if( this._totalMark>=50 && this._totalMark<100 ) {
          this._memberType = PLATINUM_MEMBER;
        } else {
          this._memberType = GENERAL_MEMBER;
        }
      }

      private void setTotalMark() {
        //设定积分
        if(this._purchase.size() >= 5) {
          this._totalMark += this._purchase.size() + PLATINUM_MEMBER;
        } else {
          this._totalMark += this._purchase.size();
        }
      }
    }

    类 Purchase:
    public class Purchase {
      private Book book;    //书
      Purchase(Book book){
        this.book = book;
      }

      public Book getBook() {
        return this.book;
      }

      private double getCharge(double priceAmount) {
        priceAmount += (float)getBook().getBookPrice() * getBook()
                    .getDiscount();
        return priceAmount;
        }
    }
```

13.3.4　更改方法特征符

修改程序后会发现类 Member 中有一个错误，这是因为 getCharge 的类型是私有型（Private）的，不能够被访问，可以用 Eclipse 重构工具中的“更改方法特征符（G）...”来解决这个问题。方法是：选中类 Purchase 中的方法“getCharge”→右击鼠标→“重构

（T）”→“移动（V）...”，如图 13-8 所示。



图 13-8　更改方法特征符

单击后出现“更改方法特征符”的窗口，如图 13-9 所示。在该窗口中可以修改方法的特征符。这里修改方法的访问修饰符，选中“public”，然后，单击“确定”。

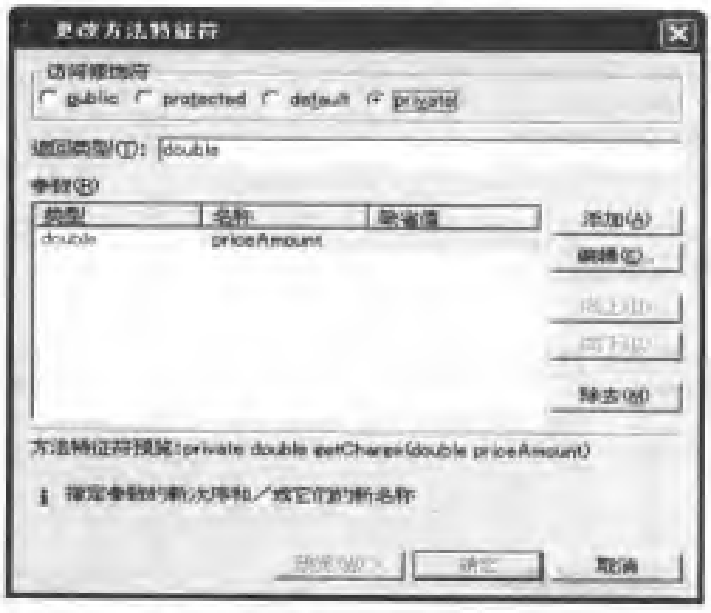


图 13-9　“更改方法特征符”的窗口

修改后的程序如下：

```
public class Purchase {
  private Book book;    //书
```

```
  Purchase(Book book){
    this.book = book;
  }

  public Book getBook() {
    return this.book;
  }

  public double getCharge(double priceAmount) {
    priceAmount += (float)getBook().getBookPrice() *
getBook().getDiscount();
    return priceAmount;
  }
}
```

13.3.5　内联

看一下方法 getCharge 中的 getBook()：在类 Purchase 中 getBook 的作用是返回 Book，从而使别的程序能够调用 Book 中的处理。但是，在类 Purchase 本身并不需要这种调用，直接使用属性 book 就可以了。可以使用 Eclipse 重构工具中的“内联（I）...”来解决这种问题。方法是：选中类 Purchase 中的方法“getBook”→右击鼠标→“重构（T）”→“内联（I）...”，如图 13-10 所示。



图 13-10　内联

然后，会弹出“内联方法”的窗口，如图 13-11 所示，在该窗口中有如下两个选项。

- 所有调用——所有调用处都内联。它含有删除方法声明，内联后删除方法。
- 仅所选调用——只限于对用户当前所选的调用内联。



图 13-11 “内联方法”的窗口

在这里选择“所有调用”，然后，单击“确定”按钮即可。修改后的程序如下：

```
public class Purchase {
  private Book book;    //书

  Purchase(Book book){
    this.book = book;
  }

  public Book getBook() {
    return this.book;
  }

  public double getCharge(double priceAmount) {
    priceAmount += (float)this.book.getBookPrice() * this.book
                .getDiscount());
    return priceAmount;
  }
}
```

用以上同样的方法，将方法 getCharge 移动到类 Book 中。在这次“移动”过程中，可能发生如图 13-12 所示的错误。

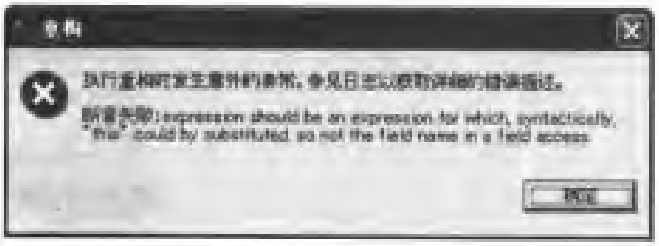


图 13-12 重构时发生错误

这是因为在方法 getCharge 中有 this 调用。将 this 去掉后再进行“移动”，问题就解决了。修改后的程序如下：

```
public class Purchase {
  private Book book;    //书

  Purchase(Book book){
    this.book = book;
```

```
    }

    public Book getBook() {
      return this.book;
    }

    public double getCharge(double priceAmount) {
      return book.getCharge(this, priceAmount);
    }
  }

  public class Book {
    private String bookName = "";  //书名
    private float bookPrice = 0;  //书价
    private float discount = 0.0f;//折扣

    public Book(String pBookName, float pBookPrice, float pDiscount) {
      bookName = pBookName;
      bookPrice = pBookPrice;
      discount = pDiscount;
    }

    public String getBookName() {
      return this.bookName;
    }

    public float getBookPrice() {
      return this.bookPrice;
    }

    public float getDiscount() {
      return this.discount;
    }

    public double getCharge(Purchase purchase, double priceAmount) {
      priceAmount += (float)getBookPrice() * getDiscount();
      return priceAmount;
    }
  }
```

13.3.6　抽取局部变量

现在，看类 Member 中的程序，如下所示。

```
//循环取出本次购买的书籍
while(iter.hasNext()) {
  purchase = (Purchase)iter.next();
  switch(this._memberType) {
    case GOLD_MEMBER:
      // 黄金会员
      priceAmount = getCharge(priceAmount, purchase) * 0.8f;
      break;
    case PLATINUM_MEMBER:
      //白金会员
      priceAmount = getCharge(priceAmount, purchase) * 0.9f;
      break;
    case GENERAL_MEMBER:
      //一般会员
      priceAmount = getCharge(priceAmount, purchase);
```

```
            break;
        }
    }
```

在这段程序中："priceAmount = getCharge(priceAmount, purchase)" 重复出现三次，一般看到这样重复出现的代码，应想办法将其合到一起。

首先，用 Eclipse 重构工具中的"抽取局部变量（V）..."，将 0.8f 抽取成局部变量 secondDiscount，方法是：选中这段程序→右击鼠标→"重构（T）"→"抽取局部变量（V）…"，如图 13-13 所示。



图 13-13　抽取局部变量

在弹出的"抽取局部变量"窗口中，填写变量名 secondDiscount，如图 13-14 所示。

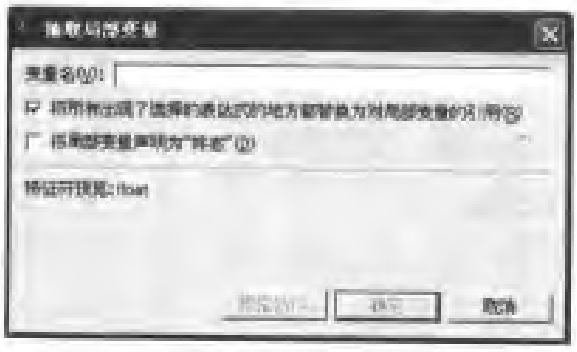


图 13-14　"抽取局部变量"窗口

单击"确定"后的程序如下：

```
public class Member {
  public double calculateBooksPrice() {

    //循环取出本次购买的书籍
```

```
        while(iter.hasNext()) {
          purchase = (Purchase)iter.next();
          float secondDiscount = 0.8f;
            switch(this._memberType) {
            case GOLD_MEMBER:
              // 黄金会员
              priceAmount = getCharge(priceAmount, purchase) * secondDiscount;
              break;
            case PLATINUM_MEMBER:
              //白金会员
              priceAmount = getCharge(priceAmount, purchase) * 0.9f;
              break;
            case GENERAL_MEMBER:
              //一般会员
              priceAmount = getCharge(priceAmount, purchase);
              break;
          }
        }

        return priceAmount;
      }
    }
```

13.3.7　重构撤销与重做

读者会发现 13.3.6 节中进行的“抽取局部变量”的重构并不成功，所以，需要撤销这一步的重构，方法是：“重构（T）...”→“撤销—抽取局部变量”，如图 13-15 所示。程序就会退回到这一步重构之前的状态。如果这时想要重新执行这一步重构，只需：“重构（T）...”→“重做—抽取局部变量”，就可以了。

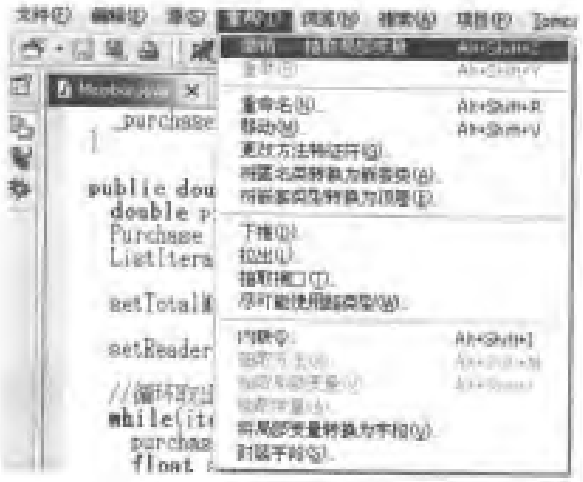


图 13-15　重构撤销

13.3.8　抽取接口

现在来重新考虑刚才的问题，对于会员，应该有黄金会员、白金会员、一般会员三

种。那么，类 Member 下应该有三个子类（subclass）才对，但是，再现在的程序中，并没有这三个子类，而是将这三个类根据不同情况将处理都放在了类 Member 中。这对将来的修改是不利的，下面重构。

引入 State 模式（由 Erich Gamana 等著的《设计模式》一书中有详细介绍），首先，生成三个类：GoldMember，PlatinumMember，GeneralMember。程序如下：

```
public class GoldMember{

  public double getCharge(double priceAmount, Purchase purchase) {
    priceAmount += getCharge(priceAmount, purchase) * 0.8f;
    return priceAmount;
  }
}

public class PlatinumMember {
      public double getCharge(double priceAmount, Purchase purchase) {
            priceAmount += getCharge(priceAmount, purchase) * 0.9f;
            return priceAmount;
      }
}

public class GeneralMember {
  public double getCharge(double priceAmount, Purchase purchase) {
    priceAmount += getCharge(priceAmount, purchase);
    return priceAmount;
  }
}
```

然后，用 Eclipse 重构工具中的“抽取接口（T）”来生成接口 BookPrice。方法是：选中类名“GoldMember”→右击鼠标→“重构（T）”→“抽取接口（T）...”，如图 13-16 所示。



图 13-16　抽取接口

在出现的“抽取接口”窗口中,填写接口名称并选择要抽取的方法。单击“确认”后，接口 BookPrice 就生成了。然后，对类 Member 进行重构，重构后的代码如下：

```
public class Member {

  public static final int GOLD_MEMBER = 1;        //黄金会员
  public static final int PLATINUM_MEMBER = 2;     //白金会员
  public static final int GENERAL_MEMBER = 3;      //一般会员

  private String _memberName = "";                 //会员名
  private int _memberType = GENERAL_MEMBER;        //会员类型
  private int _totalMark = 0;                      //积分
  private ArrayList _purchase = new ArrayList();   //购买
  private BookPrice bookPrice;

  public Member(String memberName) {

    this._memberName = memberName;

      // set total mark
      setTotalMark();

      // set member type
      setReaderYype();

      // set price code
      setPriceCode();
  }

  public void addPurchase(Purchase purchase) {
    _purchase.add(purchase);
  }

  public double calculateBooksPrice() {
    double priceAmount = 0;  //购买总金额
    Purchase purchase = null;  //当前购买
    ListIterator iter = this._purchase.listIterator();

    //循环取出本次购买的书籍
    while(iter.hasNext()) {
      purchase = (Purchase)iter.next();

    priceAmount = bookPrice.getCharge(priceAmount, purchase);
    }

    return priceAmount;
  }

  private void setPriceCode() {
      switch(this._memberType) {
        case GOLD_MEMBER:
      // 黄金会员
          bookPrice = new GoldMember();
          break;
        case PLATINUM_MEMBER:
          //白金会员
          bookPrice = new PlatinumMember();
          break;
        case GENERAL_MEMBER:
          //一般会员
          bookPrice = new GeneralMember();
```

```
            break;
        }
    }

    private void setReaderType() {
        //设定会员类型：黄金会员：1，白金会员：2，一般会员：3
        if( this._totalMark>=100 ) {
            this._memberType = GOLD_MEMBER;
        } else if( this._totalMark>=50 && this._totalMark<100 ) {
            this._memberType = PLATINUM_MEMBER;
        } else {
            this._memberType = GENERAL_MEMBER;
        }
    }

    private void setTotalMark() {
        //设定积分
        if(this._purchase.size() >= 5) {
            this._totalMark += this._purchase.size() + PLATINUM_MEMBER;
        } else {
            this._totalMark += this._purchase.size();
        }
    }
}
```

13.3.9 重命名

如果认为将 GoldMember，PlatinumMember，GeneralMember 改名为 GoldPrice，PlatinumPrice，GeneralPrice 更为贴切，则可以使用 Eclipse 重构工具中的“重命名(M)...”，以 GoldMember 为例，方法是：选中类名“GoldMember”→右击鼠标→“重构（T）”→“重命名（M）...”，如图 13-17 所示。



图 13-17 重命名

在弹出的“重命名类型”窗口中填写要更改的类名，并选择要执行的其他动作，如图 13-18 所示。

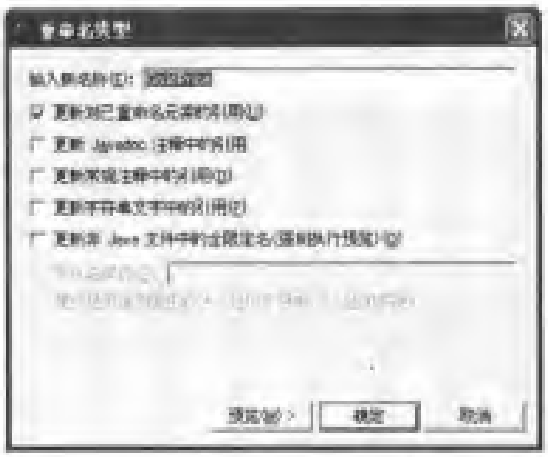


图 13-18　“重命名类型”窗口

单击“确定”后的程序如下：

```
public class GoldPrice implements BookPrice {
  public double getCharge(double priceAmount, Purchase purchase) {
    priceAmount += getCharge(priceAmount, purchase) * 0.8f;
    return priceAmount;
  }
}
```

类 Member 中调用 GoldMeber 的地方也被修改：

```
private void setPriceCode() {
    switch(this._memberType) {
     case GOLD_MEMBER:
// 黄金会员
        bookPrice = new GoldPrice();
        break;
      case PLATINUM_MEMBER:
        //白金会员
        bookPrice = new PlatinumMember();
        break;
      case GENERAL_MEMBER:
        //一般会员
        bookPrice = new GeneralMember();
        break;
    }
}
```

用相同的方法修改类 PlatinumMember 和类 GeneralMember。

13.3.10　拉出与下推

实现 State 模式也可以不使用接口，而用类的形式。现在，把程序退回到 13.3.9 节

以前的状态，程序如下：

```
public abstract class BookPrice {
  public abstract double getCharge(double priceAmount, Purchase purchase);
}

public class GoldPrice extends BookPrice{
  public double getCharge(double priceAmount, Purchase purchase) {
    priceAmount += getCharge(priceAmount, purchase) * 0.8f;
    return priceAmount;
  }
}

public class PlatinumPrice extends BookPrice{
  public double getCharge(double priceAmount, Purchase purchase) {
    priceAmount += getCharge(priceAmount, purchase) * 0.9f;
    return priceAmount;
  }
}

public class GeneralPrice extends BookPrice{
  public double getCharge(double priceAmount, Purchase purchase) {
    priceAmount += getCharge(priceAmount, purchase);
    return priceAmount;
  }
}
```

类 Member 的情况与采用接口的程序相同：

```
public class Member {
  public static final int GOLD_MEMBER = 1;        //黄金会员
  public static final int PLATINUM_MEMBER = 2;    //白金会员
  public static final int GENERAL_MEMBER = 3;     //一般会员
  private String _memberName = "";                //会员名
  private int _memberType = GENERAL_MEMBER;       //会员类型
  private int _totalMark = 0;                     //积分
  private ArrayList _purchase = new ArrayList(); //购买
  private BookPrice bookPrice;

  public Member(String memberName) {

    this._memberName = memberName;

    // set total mark
    setTotalMark();

    // set member type
    setReaderYype();

    // set price code
    setPriceCode();
  }

  public void addPurchase(Purchase purchase) {
    _purchase.add(purchase);
  }

  public double calculateBooksPrice() {
    double priceAmount = 0;  //购买总金额
    Purchase purchase = null;  //当前购买
    ListIterator iter = this._purchase.listIterator();
```

```
        //循环取出本次购买的书籍
        while(iter.hasNext()) {
          purchase = (Purchase)iter.next();

          priceAmount = bookPrice.getCharge(priceAmount, purchase);
        }

        return priceAmount;
      }

      private void setPriceCode() {
        switch(this._memberType) {
          case GOLD_MEMBER:
            // 黄金会员
            bookPrice = new GoldMember();
            break;
          case PLATINUM_MEMBER:
            //白金会员
            bookPrice = new PlatinumMember();
            break;
          case GENERAL_MEMBER:
            //  般会员
            bookPrice = new GeneralMember();
            break;
          }
        }

      private void setReaderYype() {
        //设定会员类型:黄金会员为 1,白金会员为 2,一般会员为 3
        if( this._totalMark>=100 ) {
          this._memberType = GOLD_MEMBER;
        } else if( this._totalMark>=50 && this._totalMark<100 ) {
          this._memberType = PLATINUM_MEMBER;
        } else {
          this._memberType = GENERAL_MEMBER;
        }
      }

      private void setTotalMark() {
        //设定积分
        if(this._purchase.size() >= 5) {
          this._totalMark += this._purchase.size() + PLATINUM_MEMBER;
        } else {
          this._totalMark += this._purchase.size();
        }
      }
    }
```

现在，要添加设定礼券的方法，如果会员一次性购书金额在 100 元以上，那么就奖励该会员一张礼券，礼券可以在购买其他商品时使用。

在类 GoldPrice，PlatinumPrice，GeneralPrice 中分别添加计算礼券的方法 setGifttoken。修改后的三个程序如下：

```
public class GoldPrice extends BookPrice{

  public double getCharge(double priceAmount, Purchase purchase) {
    priceAmount += getCharge(priceAmount, purchase) * 0.8f;
    return priceAmount;
  }
```

```
    public double getGifttokenCnt() {
      return 1;
    }
  }

  public class PlatinumPrice extends BookPrice{
    public double getCharge(double priceAmount, Purchase purchase) {
      priceAmount += getCharge(priceAmount, purchase) * 0.9f;
      return priceAmount;
    }

    public double getGifttokenCnt() {
       return 1;
    }
  }

  public class GeneralPrice extends BookPrice{
    public double getCharge(double priceAmount, Purchase purchase) {
      priceAmount += getCharge(priceAmount, purchase);
      return priceAmount;
    }

    public double getGifttokenCnt() {
      return 1;
    }
  }
```

可以看到类 PlatinumPrice、GeneralPrice 和 GoldPrice 的 getGifttokenCnt 方法是一致的，可以将该方法移到它们的父类 BookPrice 中，这个处理通过 Eclipse 重构工具中的“拉出（W）...”来完成。方法是：选中 GeneralPrice 中的方法“getGifttokenCnt”→右击鼠标→“重构（T）”→“拉出（L）...”，如图 13-19 所示。



图 13-19　拉出

在“拉出”窗口中选择想要放到类 BookPrice 中的方法，如图 13-20 所示。

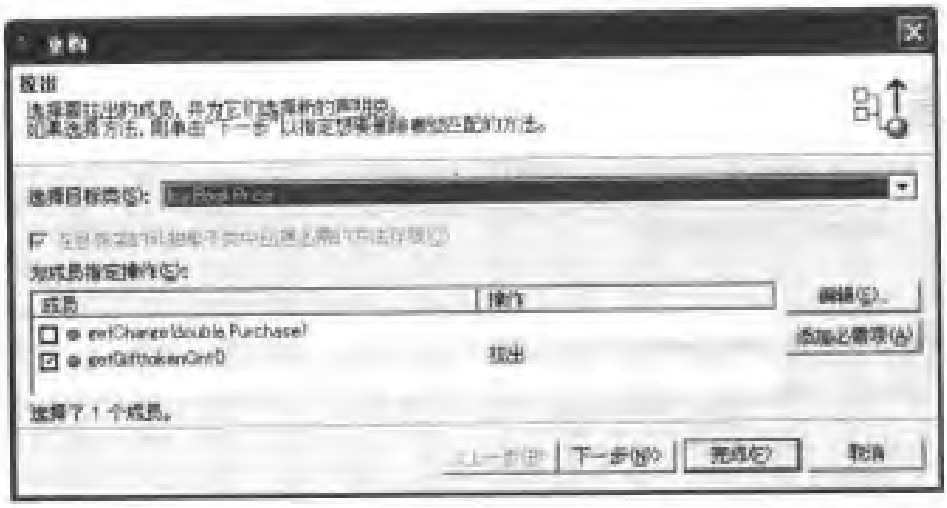


图 13-20　选择想要放到类 BookPrice 中的方法

单击“下一步（N）”，在出现的窗口中选择其他的类，如图 13-21 所示。这里，暂且只选择类 GeneralPrice 和类 PlatinumPrice（因为它们对礼券数量的计算公式是相同的）。

选择“下一步（N）”后，单击“完成（F）”。

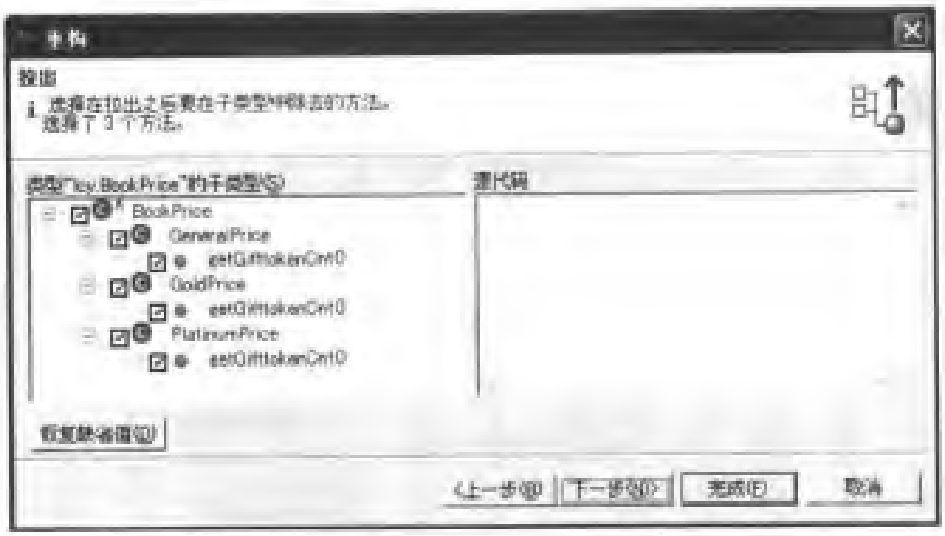


图 13-21　选择其他的类

修改后的程序如下：

```
public class GoldPrice extends BookPrice{

  public double getCharge(double priceAmount, Purchase purchase) {
    priceAmount += getCharge(priceAmount, purchase) * 0.8f;
    return priceAmount;
  }
}
```

```
public class PlatinumPrice extends BookPrice{
  public double getCharge(double priceAmount, Purchase purchase) {
    priceAmount += getCharge(priceAmount, purchase) * 0.9f;
    return priceAmount;
  }
}

public class GeneralPrice extends BookPrice{
  public double getCharge(double priceAmount, Purchase purchase) {
    priceAmount += getCharge(priceAmount, purchase);
    return priceAmount;
  }
}

public abstract class BookPrice {

  public abstract double getCharge(double priceAmount, Purchase purchase);

  public double getGifttokenCnt() {
    return 1;
  }
}
```

这时，如果想将方法 getGifttokenCnt 放到三个子类中，可以使用 Eclipse 重构工具中的“下推（D）...”，方法是：在类 BookPrice 中选中方法 getGifttokenCnt→右击鼠标→“重构（T）”→“下推（D）...”，如图 13-22 所示。



图 13-22 下推

在出现的“下推”的窗口中单击“确定”后，“下推”动作就完成了。修改后的代码与“拉出”前的代码一样。

本章小结

对例子程序的重构工作就做到这里，通过这个例子相信读者已经能够掌握 Eclipse 重构工具中的 14 种重构手法（包括重构撤销和重构重做）。希望大家不但从中学会使用 Eclipse 重构工具，而且能够对“重构”这项技术有所了解。要想更好地掌握重构，仅阅读本章内容是不够的，还需要参看其他有关书籍，如 13.1 节中推荐的图书，并多实践，在工作中细心体会、积累经验。

第 14 章　Eclipse 插件开发

Eclipse 最具魅力的地方就是插件，它是 Eclipse 可以与其他工具无缝集成的关键。Eclipse 在启动的时候要完成的工作只有两个：启动平台的基础部分和查找系统的插件。在 Eclipse 中绝大部分功能的实现是由相应的插件完成的，比如 WrokBench UI 插件完成界面的外观显示，Resource Management 插件完成维护、生成项目或文件等资源的管理工作，等等。虽然许多功能都是绝大多数 IDE 环境所必备的功能，Eclipse 却都把它们都做成了插件模式，连用来开发 Java 程序的开发环境（Java development tooling，JDT）也只不过是 Eclipse 系统中的一个普通插件而已。整个 Eclipse 体系结构就象一个大插板，可以不断的添加插件，同时，还可以在现有插件上再添加插件。本章简单介绍了插件，以及怎样开发一个插件。具体内容包括：

- Eclipse 基于平台的体系结构
- 插件介绍
- 插件开发环境（PDE）的配置及使用 PDE 创建插件
- 调试插件
- 部署插件
- 运行插件

14.1　Eclipse 平台体系结构

Eclipse 平台是根据扩展点的概念来建立的。扩展点是系统中严格定义的一些位置，而其他工具（称为插件）可以在此添加功能。平台中的每个主要子系统本身都是由一组插件来构成的，这些插件实现一些重要功能并定义扩展点。Eclipse 系统本身就是通过向第三方插件供应商提供可以使用的相同扩展点来构建的。插件可以定义它们自己的扩展点，或者只是将扩展添加至其他插件的扩展点。

平台子系统通常将可视功能部件添加到平台中，并提供 API 来扩展它们的功能。这些可视功能部件中的某些组件与扩展点不直接相关，但可以用来实现扩展附加类库。例如，工作台用户界面提供 JFace 用户界面框架和 SWT 小窗口工具箱。

Eclipse SDK 中包含基本平台以及以下两个主要工具，它们对于插件开发很有用。

- “Java 开发工具（JDT）”：通过提供用于编辑、查看、编译、调试和运行 Java 代码的专门功能部件来扩展平台工作台。JDT 是作为包括在 SDK 中的一组插件安装的，实现了 Java 开发环境的全部功能。
- “插件开发环境（PDE）”：提供用来自动创建、处理、调试和部署插件的工具。PDE

是作为包含在 SDK 中的一组插件来安装的，并且添加了专门工具使开发和扩展插件变得更为方便。

这些工具不仅有用，而且还提供许多示例来说明如何通过构建用来扩展系统的插件将新工具添加到平台中（参见图 2-1）。

Eclipse 启动时，由“运行时”负责在 plugin.xml 文件中查找插件的声明，每个 plugin.xml 文件都在各插件的子目录中。这些子目录位于 Eclipse 的安装目录下名为 plugins 的公共目录中。根据这些文件，运行时在内存中构建一个全局注册表，称为插件注册表，根据这个注册表，给定的插件就可以在运行时确定其他哪些插件希望扩展它。希望被其他插件扩展的插件将声明一个扩展点，而其他插件可以通过对这个插件声明的扩展来，利用这个插件。

14.2　插件简介

Eclipse 平台是在其上创建插件的核心框架和服务的名称。它提供了在其中装入和运行插件的运行时。插件通过添加到预定义的扩展点向平台添加功能。工作台用户界面就是由一个插件添加的。启动工作台时，用户并未启动单个 Java 程序。在激活平台运行时的时候，它可以动态地发现已注册的插件并根据需要来启动它们。

当用户想提供用来扩展平台的代码时，可通过在插件中定义系统扩展来完成此操作。平台具有一组严格定义的扩展点——可在扩展点处进入平台并添加系统行为。在平台的透视图中，用户的插件与基本插件如资源管理系统或工作台本身一样，没有任何不同。

如果想把用户的代码变为插件，可以按照如下步骤进行：

- 决定用户的插件将如何与平台集成。
- 标识需要进行添加的扩展点以便与用户的插件进行集成。
- 根据扩展点的规范来实现这些扩展。
- 提供清单文件（plugin.xml），它描述用户正在提供的扩展以及代码的封装。

将经典的“Hello World”作为插件实现，是如何创建插件的最好说明。在 14.3 节中，将会实现它。实现的时候会略过一些详细信息，以便让它可以运行。

下面介绍几个基本概念。

1．功能部件

功能部件以包含它们的插件的形式来描述。这意味着插件是封装功能的基本单位。组织功能部件的目的是分发和更新产品，而组织插件的目的是为了在产品开发小组中促进产品功能的开发。产品开发小组决定何时将程序功能分割为单独的插件。插件封装在插件归档文件中，并使用插件清单文件 plugin.xml 来描述。

2．插件片段

插件片段是单独封装的文件，对其内容的处理就如同它们是在原始插件归档文件中一样。它们对于添加插件功能很有用，例如，在其安装之后将附加的本地语言转换为现有插件。插件片段可以在不形成一个新的发行版本的情况下，以增量的形式将功能部件

添加到现有插件上，因为使用它们来添加功能部件，无需重新封装或重新安装原始插件。当平台检测到片段时，将把其内容与原始插件中的功能合并。换句话说，若查询平台的插件注册表，将看到片段中提供的扩展和其他功能就如同它在原始插件中一样。片段使用片段清单文件 fragment.xml 来描述。它与插件清单文件相似。因为片段与其插件共享许多属性，插件清单中的一些属性无效，例如，插件类和插件导入。插件归档文件可以包含插件或片段。

14.3 开发一个简单的视图插件 HelloWorld

本节将介绍开发一个简单视图插件的例子。

14.3.1 创建插件

创建插件可参考以下具体步骤：

（1）打开“新建项目”向导：“文件（F）”→“新建（N）”→“项目（R）”，并从插件开发类别中选择插件项目。单击“下一步（N）”，如图 14-1 所示。

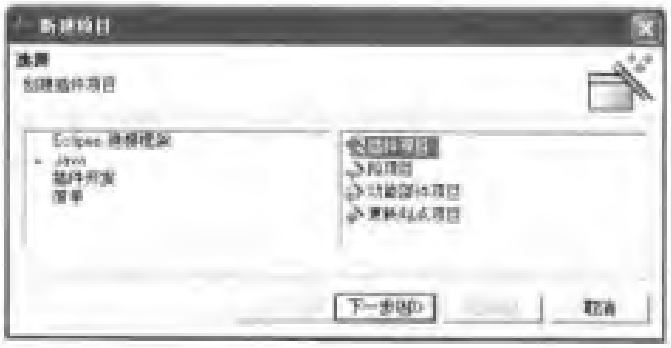


图 14-1 新建插件项目

（2）“新建插件项目”向导中以 com.myPlugin.examples.helloworld 作为项目的名称，这个名称应当是惟一的。单击“下一步（N）”，如图 14-2 所示。

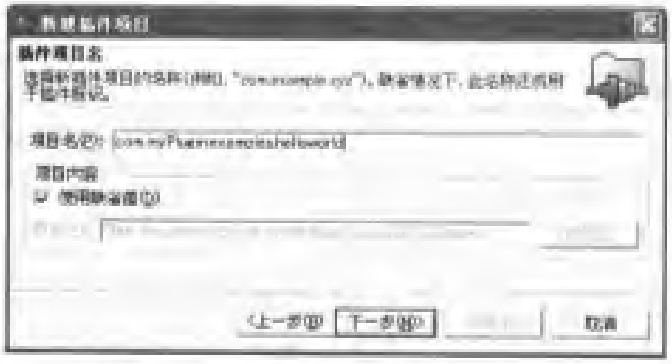


图 14-2 指定项目名

（3）“新建插件项目”向导中，保持默认选项。单击“下一步（N）”，如图 14-3 所示。



图 14-3　保持默认选项

（4）“新建插件项目”向导中，选择“创建空白插件项目（C）”。然后单击“完成（F）”，如图 14-4 所示。

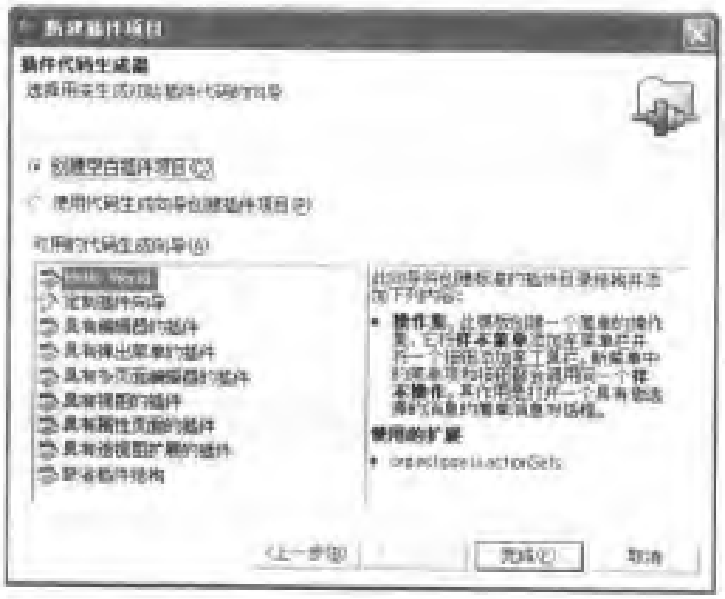


图 14-4　创建空白插件项目

（5）当出现询问用户是否想切换到“插件开发”透视图时，选择“是（Y）”，如图 14-5 所示。

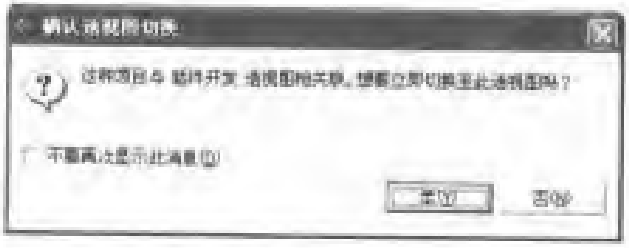


图 14-5　选择切换到“插件开发”透视图

（6）插件做好后，Eclipse 平台如图 14-6 所示。



图 14-6　Eclipse 平台

（7）右击 src 目录，选择“新建（N）”→“包”，名称中填写“com.myPlugin.examples.helloworld”。单击“完成（F）”，如图 14-7 所示。



图 14-7　新建 com.myPlugin.examples.helloworld 包

（8）右击新建立的包，选择“新建（N）”→“类”，名称中填写“HelloWorldView”。单击“完成（F）”，如图 14-8 所示。



图 14-8　新建 HelloWorldView 类

（9）编辑 HelloWorldView.java 类

在包资源管理器中打开新建立的类 HelloWorldView.java，编辑类 HelloWorldView。代码如下：

```
package org.eclipse.examples.helloworld;

import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.SWT;
import org.eclipse.ui.part.ViewPart;

public class HelloWorldView extends ViewPart {
    Label label;
    public HelloWorldView() {
    }
    public void createPartControl(Composite parent) {
        label = new Label(parent, SWT.WRAP);
        label.setText("Hello World");
    }
    public void setFocus() {
    }
}
```

（10）保存编辑后的类，Eclipse 工作台会提示用户“无法解析 ViewPart 和 SWT.WRAP”，还需要导入插件 org.eclipse.ui。首先，打开导入向导：“文件（F）”→“导入（I）”，在导入向导中选择“外部的插件和段”。单击“下一步（N）”，如图 14-9 所示。



图 14-9　导入外部插件和段

（11）在“导入插件和段”向导中按照默认选项单击“下一步（N）”直到出现如图 14-10 所示画面。将“org.eclipse.swt”和“org.eclipse.ui”选中，单击“完成（F）”。



图 14-10　导入 org.eclipse.swt 和 org.eclipse.ui

（12）可以看到包资源管理器中多出两个项目“org.eclipse.swt”和“org.eclipse.ui”，如图 14-11 所示。

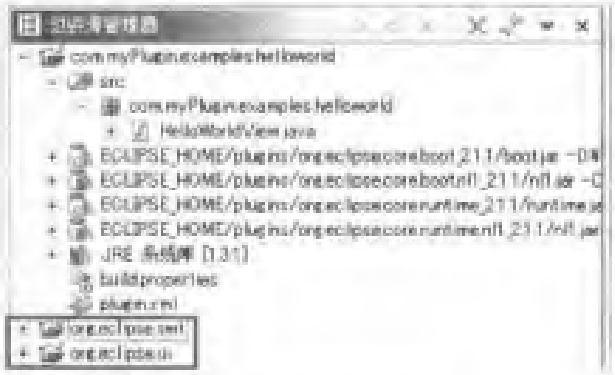


图 14-11　包资源管理器

现在需要将这两个项目导入到刚建立的 helloworld 插件项目中。方法是右击“com.myPlugin.examples.helloworld”项目，选择“属性（R）”。在弹出的向导中，单击左侧的“Java 构建路径”。在右侧的选项中选择“项目（P）”选项卡，将“org.eclipse.swt”和“org.eclipse.ui”选中，单击“确定”，如图 14-12 所示。



图 14-12　导入项目 org.eclipse.swt 和 org.eclipse.ui

可以看到刚才 Eclipse 工作台所提示的无法解析 ViewPart 和 SWT.WRAP 等错误信息会消失，这是因为 Eclipse 工作台会自动重新编译程序。如果 Eclipse 工作台没有重新编译，那么可以选择“项目（P）”→“全部重建（L）”，来重建项目。

还需要修改 plugin.xml 文件，才能将刚才做的这个视图添加至工作台。在包资源管理器中双击 plugin.xml，打开它的源代码，将如下代码替换它的源代码。

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
   id="com.myPlugin.examples.helloworld"
   name="Hello World Example"
   version="1.0.0">
   <requires>
      <import plugin="org.eclipse.ui" />
   </requires>
   <runtime>
```

```
        <library name="HelloworldPlugin.jar"/>
    </runtime>
    <extension point="org.eclipse.ui.views">
        <category id="com.myPlugin.examples.helloworld.hello" name="Hello" />
        <view id="com.myPlugin.examples.helloworld.HelloWorldView"
            name="Hello Greetings"
            category="com.myPlugin.examples.helloWorld.hello"
            class="com.myPlugin.examples.helloworld.HelloWorldView" />
    </extension>
</plugin>
```

下面介绍这个文件中的各标签的属性。

- <plugin>标签：该标签的属性提供的是要开发的 Welcome 插件的基本信息。属性 name，id 和 version 分别声明了这个插件的名称，标识和版本。其中 id 属性被要求不能和现有的 Eclipse 插件的 id 有冲突。插件的名称是“Hello World Example”。标识是 com.myPlugin.examples.helloworld，注意，这个标示必须是惟一的。其版本为 1.0.0 版。
- <requires>标签：该标签声明了必需的插件，由于在插件中使用了工作台和 SWT，因此必须导入 org.eclipse.ui。
- <extension>标签：该标签声明了这个插件是从哪个扩展点上扩展的。该标签有几个不同的配置参数。通过 category 为视图声明类型。在 Eclipse 工作台的显示视图向导中，可以使用类型来将相关视图集中在一起。在上面的代码中定义的类型名为“Hello”，以便使它显示在它自己的组中。还为视图声明了惟一的标识，并指定了用来实现视图的类的名称。还为视图指定了名称“Hello Greetings”，它将显示在“显示视图”向导和视图的标题栏中。

14.3.2 安装和运行插件

通过上面的过程已经将插件创建好了，下面把这个插件安装到 Eclipse 中。

（1）首先将插件做成 JAR 文件。选择“文件（F）”→“导出（O）”。在导出向导中选择“JAR 文件”，单击“下一步（N）”，如图 14-13 所示。

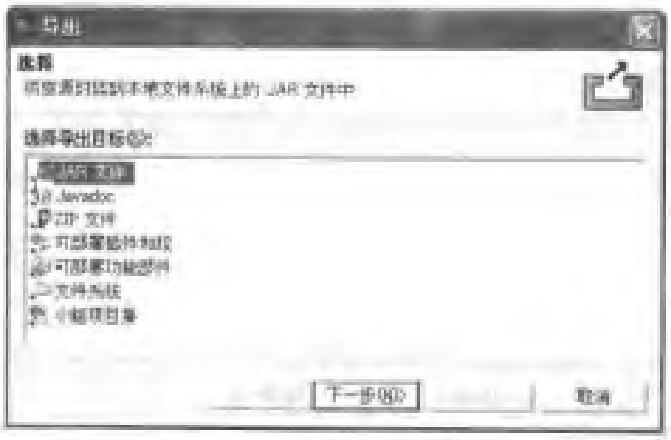


图 14-13 导出项目

在“JAR 导出”向导中，单击“浏览（R）”按钮，选择保存 JAR 文件的路径。新建一个“com.myPlugin.examples.helloworld”的文件夹，把它保存到 eclipse 安装路径下的 plugins 文件夹下，文件名为“HelloworldPlugin.jar”。单击“完成（F）”，生成 JAR 文件，如图 14-14 所示。



图 14-14　以 JAR 文件的形式导出项目

（2）JAR 文件导出完成后，还有一个重要的“plugin.xml”文件需要导出到“com.myPlugin.examples.helloworld”文件夹下。plugin.xml 文件提供注册扩展，该清单文件主要用于描述插件，包括插件的代码所在的位置以及正在添加的扩展。在包资源管理器中，右击 plugin.xml 文件，选择“导出（X）”，在导出向导中选择“文件系统”，然后单击“下一步（N）”，如图 14-15 所示。

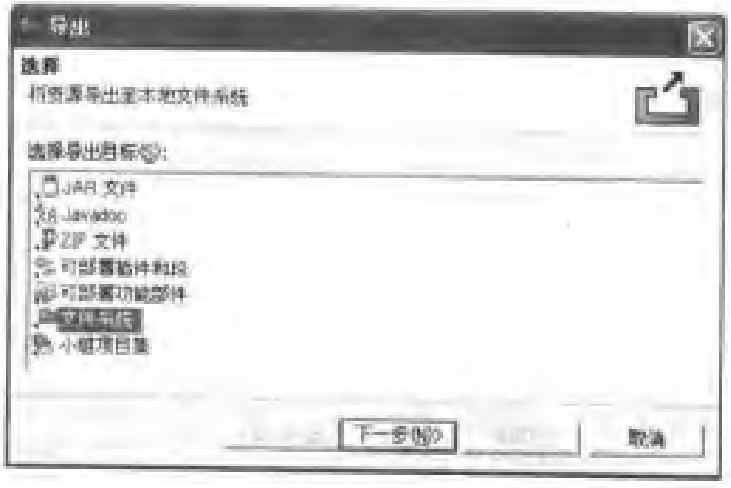


图 14-15　导出 plugin.xml

单击“至目录”项的“浏览（R）”按钮，将文件保存到刚刚新建的“com.myPlugin.examples.helloworld”文件夹中。单击“完成（F）”，如图 14-16 所示。



图 14-16　导出至项目目录

（3）如果当前 Eclipse 工作台正在运行，则需要先关闭然后再重新启动。当平台启动时，它将装入系统中安装的所有插件的列表，称为插件注册表。注册表记录插件和它们进行添加的扩展点。重新启动工作台后插件注册表将找到新插件。

（4）重新启动 Eclipse 工作台后，单击“窗口（W）”→“显示视图（V）”→“其它（O）”，打开显示视图向导，如图 14-17 所示。



图 14-17　显示视图向导

（5）在显示视图向导中，选择“Hello”→“Hello Greetings”视图，单击“确定”。工作台将激活插件，实例化并初始化视图类，并在工作台中显示新视图以及所有其他视图，如图 14-18 所示。

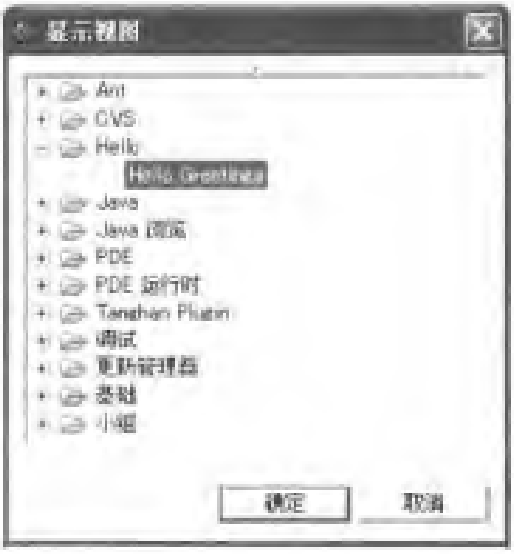


图 14-28　激活插件

（6）可以看到建立的视图被显示在 Eclipse 工作台上，如图 14-19 所示。

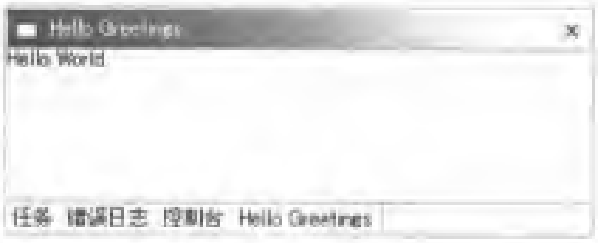


图 14-19　Hello 视图

这就是创建并运行的第一个插件。工作台用户界面就是由类似这样的若干个插件组合而成的。启动工作台时，并未启动单个 Java 程序，但在激活平台运行时的时候，它可以动态地发现已注册的插件并根据需要来启动它们。当用户想提供用来扩展平台的代码时，可以通过在插件中定义系统扩展来完成。平台具有一组严格定义的扩展点——可在扩展点处进入平台并添加系统行为。在平台的透视图中，用户的开发的插件与基本插件诸如资源管理系统或工作台本身一样没有任何不同。

14.4　插件开发环境简介

“插件开发环境”（PDE）是一个工具，在平台工作台中工作时，它可以用来帮助用户开发平台插件。PDE 提供了一组平台扩展添加项（视图、编辑器和透视图等），这些添加项得在工作台内开发插件的进程更为简便。与一般平台原理一样，PDE 不是单独启动的工具。安装 PDE 时，工作台看起来是一样的，但是用户可看到新的透视图、向导、编辑器和视图。PDE 与平台溶合在一起，并且通过新透视图来提供其功能。

PDE 在平台中采用“项目”结构方式。当使用 PDE 时，用户的工作将以项目、文件夹和文件的形式组成。PDE 查看工作区中的项目并特别注意那些将插件或段清单文件作为项目的直接子代的项目。如果一个项目具有插件清单，则 PDE 认为它是插件项目。另外，PDE 将特殊的 PDE 性质与这些项目相关联，以便能够连接特殊的 PDE 构建器。众所周知，平台是基于插件这一概念的，而插件具有明确定义的结构和规范。PDE 的目标是确保用户在创建新插件时遵循规则，以便让平台正确地合并这些插件。PDE 管理项目中的插件，每个项目都代表一个插件，并包含定义该插件的文件夹和文件，以及它将与平台交互作用的方式。这些插件在用户的工作区中，并被显示在资源导航器视图以及显示工作区资源的其他视图中（例如，“Java 包资源管理器”）。

14.4.1 主机和运行时

PDE 中最重要的概念是主机和运行时工作台实例的概念。

启动工作台时，将使用它来处理用户正在构建的插件的项目。在使用 PDE 和其他工具开发插件时，用户正在运行的工作台实例是主机实例。在此实例中可用的功能部件是来自随应用程序一起安装的插件。一旦用户对插件比较满意并且想测试它时，就可以启动另一个工作台实例，即运行时实例。此实例包含与主机实例相同的插件，还具有用户在主机实例中所开发的插件。PDE 启动器将小心地将用户的插件与主机插件合并，并创建运行时实例。

14.4.2 外部插件与工作区插件

由于运行时工作台实例显示的是由来自两个不同地方（用户当前的工作区和原始的主机安装）的插件所提供的功能部件集合，因此 PDE 会将这些插件识别为两个不同的“种类”。

1. 外部插件

外部插件是由基本平台安装提供的一种插件，只可以从它们的原始位置引用它们，而不能进行修改。用户可以引用、浏览这些插件或者查看它们的源代码以及调试它们，但是，它们被认为是只读的。

2. 工作区插件

工作区插件是主机工作台中还在开发的那些插件。它们由用户来控制，能够以任何形式添加、删除和修改。

由于外部插件和工作区插件的位置不同，因此 PDE 以不同的方式处理它们。例如，打开工作区插件的插件清单将在工作区中的该清单文件上启动编辑器。外部插件的相同操作将在引用主机安装位置中的插件清单文件的外部 URL 上启动编辑器。

虽然这些区别对于 PDE 用户是比较透明的，但是理解这些区别还是有必要的。

14.4.3　配置 PDE

尽管 PDE 是 SDK 平台的一部分，但是，在特定的安装中可能会提供也可能不提供它。如果未提供，则可以使用“更新管理器”平台来添加它。一旦安装了 PDE，它就将为它自己配置准备使用的合理的缺省值。

1. 运行时实例配置

在 PDE 中可以设置的最重要的变量就是运行时工作台实例路径。这是将用来运行和调试插件的工作台的安装路径。运行时工作台的缺省选项是主机工作台，即，被用于开发的工作台。当用户想要创建将安装到类似于主机设置的平台配置中的插件时，非常容易接受此默认值。当然，也可以对开发和运行时使用不同的工作台实例。例如，用户可能已经有了一个用于开发的稳定的平台配置，但是，用户希望将插件部署到另一个不同版本的行时工作台中。而这个新的运行时实例可能基于新技术（可能不太稳定），或者可能合并了与主机配置不同的插件集。在这种情况下，对主机和运行时平台使用不同的工作区是很重要的。主机工作区位置（在此处的所有项目都将存入硬盘）是在启动主机开发平台时设置的。要设置运行时工作区的位置，选择“窗口（W）”→“首选项（P）”对话框中“插件开发”的“目标平台”页面，如图 14-20 所示。



图 14-20　目标平台选项卡

当用户设置应用程序位置并按重新装入按钮时，页面上的插件和段的列表将填充所发现的外部插件。缺省情况下，PDE 将看不见这些插件，可以通过个别启用这些插件或者按全部选中按钮来使它们可视。

注意，PDE 将使用工作区插件和已启用的外部插件来解析 Java 类和插件依赖性。在工作区插件与外部插件具有相同标识的情况下，工作区插件将屏蔽它的外部插件。

14.4.4 PDE 提供的视图

PDE 提供了两个视图：错误日志和插件注册表。可以通过下面的步骤打开这两个视图。选择“窗口（W）”→“显示视图（V）”→“其它（O）”，在“显示视图”向导的“PDE 运行时”中有这两个视图，可以选择并打开它们，如图 14-21 所示。

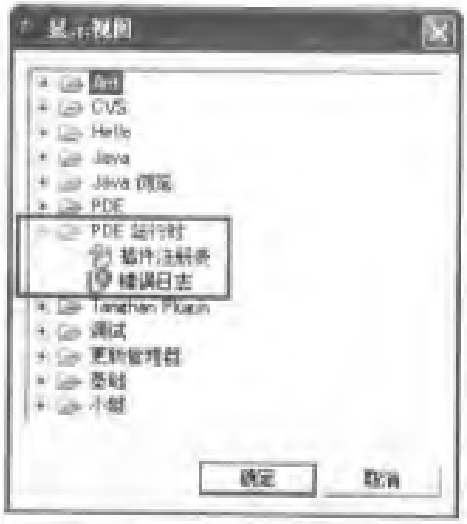


图 14-21 PDE 运行时所提供的视图

1. 错误日志视图

错误日志视图（见图 14-22）将捕获由平台或者用户的代码抛出的内部错误。这在运行时是很有用的，因为在开发期间可能会产生异常。触发错误时，会将它们添加到记录视图中，可以通过双击它们来检查详细信息，还可以在解决完错误后清除该视图。在工作区中.metadata 位置的.log 文件也存储了相同的信息，但是“错误日志”更加方便。注意，始终都可以通过重新装入.log 文件来恢复视图的内容。错误信息是按逆向年代顺序排列的（最晚的时间排在第一位），并且可以使用“属性”对话框来查看错误的详细信息。

某些错误可能是分层的（即，对于一个复杂问题可能有一个总条目，还有大量子条目，每个子条目详细显示个别问题）。

2. 插件注册表视图

插件注册表视图显示正在运行的实例的插件注册表的只读视图。此视图在运行时工

作台实例中很有用，它可以使用户了解是否已启动自己的插件以及平台能够识别哪些插件帮助文件（扩展、扩展点和库，等等）。“属性”视图中显示了“注册表”视图中所选对象的详细属性。



图 14-22　错误日志视图

缺省情况下，“注册表”不会在 PDE 透视图中显示。必须手动打开它。记住，必须是在为了进行测试而启动的实例中而不是在用于开发的实例中执行此操作。



图 14-23　插件注册表视图

14.5　使用 PDE 创建插件

简单了解 PDE 之后，下面通过创建插件来进一步了解怎样使用 PDE 开发插件。在

这个过程中会略过一些和前一个例子中重复的步骤。

14.5.1 创建插件

1. 从创建特殊插件项目开始

PDE 提供了向导来设置项目。可以使用“文件（F）”→“新建（N）”→“项目（R）”，并在向导的列表中的“插件开发”类别下面选择插件项目来访问此向导。当按“下一步”后，在“新建插件项目”向导中以“com.myPlugin.examples.helloworld2”作为项目的名称。按照惯例，插件项目名与插件标识是相同的，但是它们可以不相同。下一页显示插件项目的初始结构。如果项目名符合插件名的规则，则可以保留缺省值，否则，将要更改它。在这里不做任何改变，保持默认值，如图 14-24 所示。



图 14-24 新建插件项目

“插件运行时库（R）”的名称将用来定义一旦用户准备发布新插件的代码将在何处封装这些代码以及封装后的库文件名。“源文件夹（S）”告诉向导要创建的源文件夹的名称，并将 Java 源代码存储在新文件夹内。如果用户正在创建并不打算包含 Java 代码的插件（例如，文档插件），则选择“创建简单项目”并按“完成（F）”按钮。PDE 将创建一个包含具有最基本内容的 plug-in.xml 文件的简单项目。

2. 选择新插件的内容的外观

如果选择空白插件，则将只创建插件项目（上一个例子就是采用这种方法）。如果使用模板向导，则 PDE 还将创建一些文件，例如，插件类、构建属性以及插件清单。PDE 提供的某些模板还生成其他一些有用的内容，例如，视图、编辑器和属性页面等等。此内容是通过可组合到向导中的模板生成的。所有可用的模板都可以在“定制插件向导”中单独选择。此例中使用“Hello, World”模板向导，可以在向导列表右边的区域中阅读向导的相关信息，如图 14-25 所示。

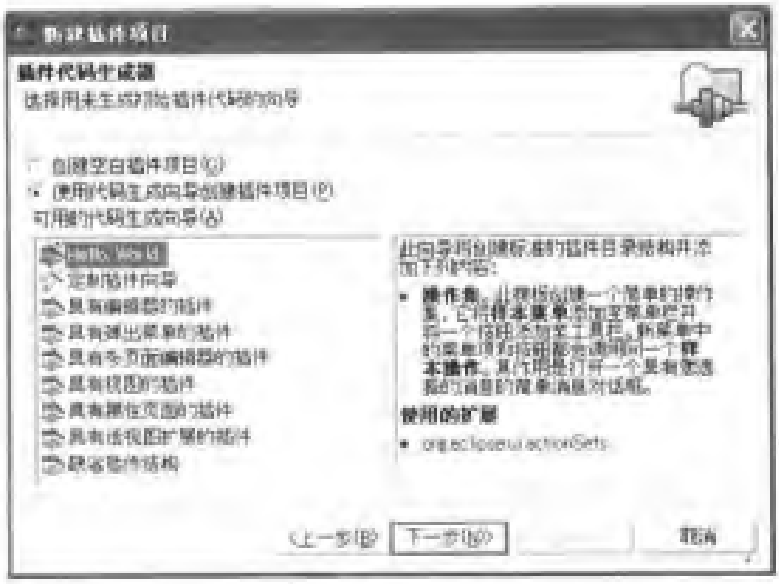


图 14-25　使用“Hello, World”模板创建插件项目

3. 完成插件的创建

在选择了“Hello, World”模板向导之后，下列页面就受到向导的控制。每个模板向导都拥有定制要生成的内容的许多页面。但是，所有模板都共享一个公共页面。它将收集有关插件的信息，例如，插件的名称、版本和供应商。它还将提供与将要生成的插件类相关的选项。在此例中，接受所有默认选项。单击“完成（F）”按钮，完成插件的创建，如图 14-26 所示。



图 14-26　新建插件项目的信息

插件创建完成后，Eclipse 工作台如图 14-27 所示。

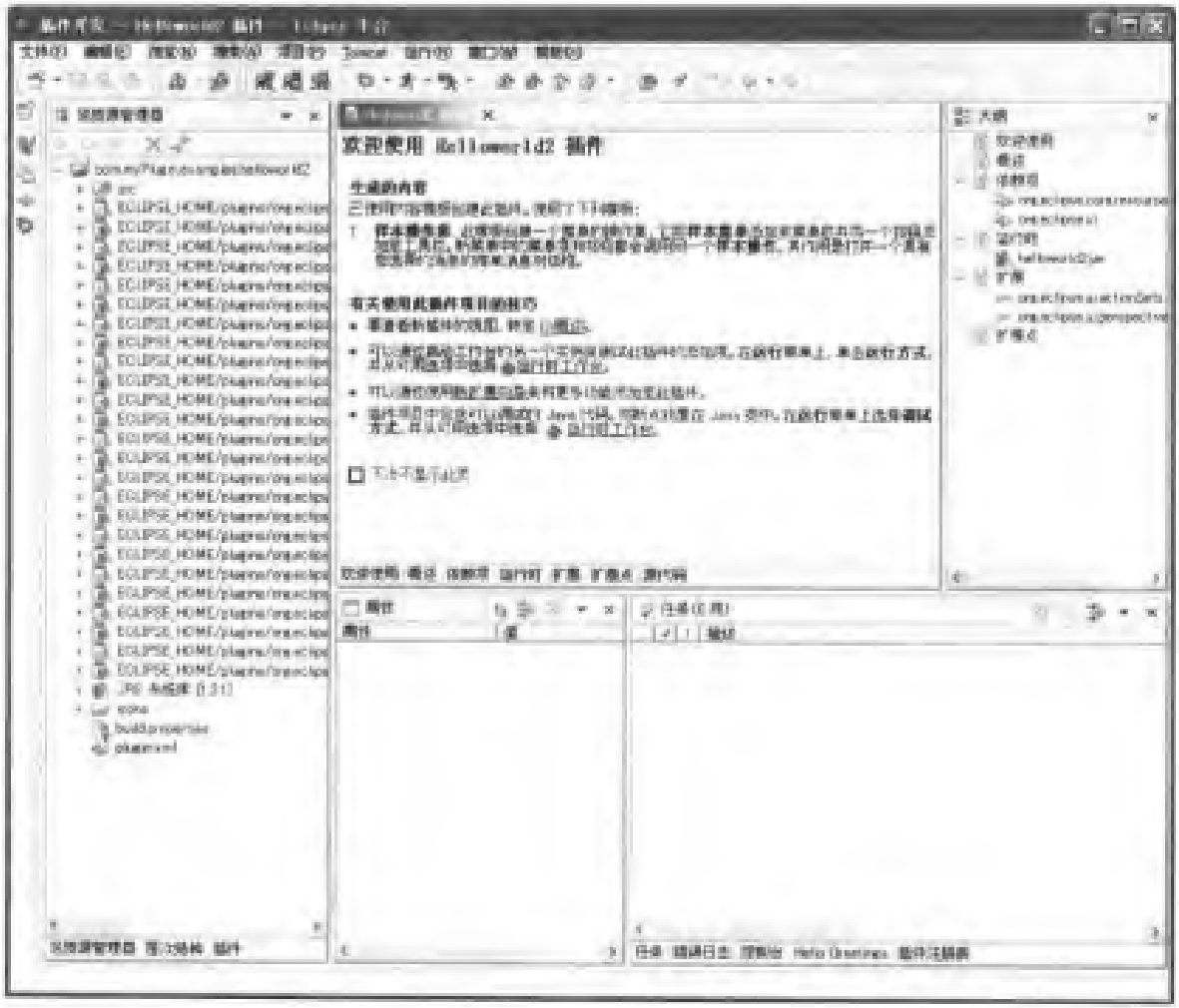


图 14-27 Eclipse 工作台

14.5.2 插件清单编辑器

在创建了 PDE 插件之后，在工作台上将看到新插件的插件清单编辑器。

PDE 具有一些多页面编辑器，包括共享同一基本行为的插件清单编辑器。该编辑器具有一个源页面，该页面显示原始文件内容，还具有一个或多个表单页面，它以更为结构化的格式来提供相同的信息。

插件清单编辑器将表单信息划分成几个页面，以便更加清晰。

1. “欢迎”页面

使用模板向导创建插件后，清单编辑器打开的默认页面是“欢迎”页面。它的用途是总结由所选择的模板生成的内容和帮助用户提高效率。如果用户很熟悉插件开发，就可以通过选择左下角的“下次不显示此页”的复选框来停止显示该页面。一旦关闭了编辑器，就会除去“欢迎”页面的信息，并且在下次打开清单进行编辑时将不会再出现该页面信息，如图 14-28 所示。

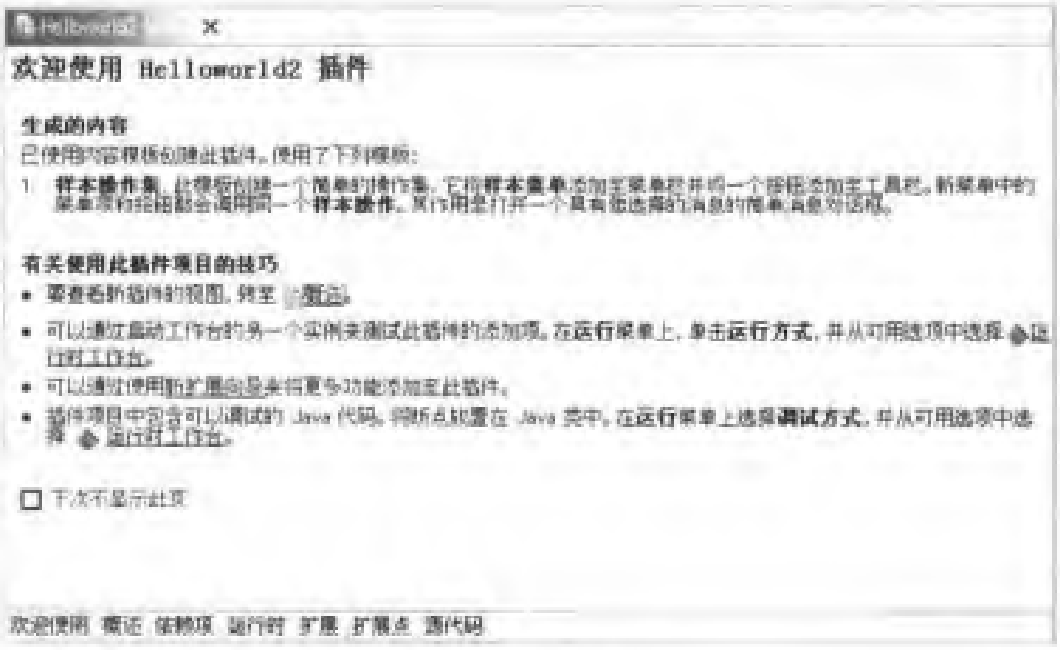


图 14-28　“欢迎”页面

2. “概述”页面

“概述”页面显示了分成几部分的“插件概览”信息。该页面上显示了每个部分的最重要信息，并提供超级链接和多个按钮，以访问其他信息，可以编辑一般插件信息。此页上的其余信息只能够浏览。如果为插件指定了插件类，则超级链接将在 Java 编辑器中打开类，如图 14-29 所示。



图 14-29　“概述”页面

警告部分将显示警告问题以及未执行的与插件相关联的任务。例如，它将标记编译

错误和警告，以及插件未解决的错误或者循环引用等问题。

3. “依赖项”页面

“依赖项”页面显示插件对其他插件的依赖性。插件当前所依赖的插件被显示在列表中。可以通过添加更多插件（使用“添加（A）”按钮）或者从列表中除去插件来修改该列表。此列表中发生的更改将提示 PDE 更新插件清单并配置项目的构建路径，以便正在开发中的插件可以看见已导入的插件的类。由于 PDE 将工作区插件和外部插件区分开来，因此，合格的插件将显示在两个单独的复选框列表中。一旦将插件引用添加到列表中，就可以通过从弹出菜单中选择打开或者双击条目来浏览它，如图 14-30 所示。



图 14-30 “依赖项”页面

当在必需的插件列表中选择一个条目时，就可以在导入详细信息部分中重新导出依赖性。通过重新导出，可以使导入它的插件看见该插件。还可以更具体地考虑版本和需求的匹配。缺省情况下，具有必需标识的任何插件都是可用的。可以通过指定版本以及是否应该显式指定版本匹配来添加约束。

可以使用右下角的部分来获取依赖性图形信息。可以从下拉组合框中选择信息的类型。可以使用该部分来检查依赖性图形中的循环，引用用户的插件的段以及需要用户插件的其他插件。在每种情况下，结果都将显示在组合框下面的列表中。

缺省情况下，依赖性列表中的更改将自动触发更新插件项目的构建路径。可以在“插件开发”→“Java 构建路径控制”首选项页面上关闭该功能部件。如果关闭了此功能部件，并且在依赖性列表中添加或者删除了插件，则应手动更新构建路径，以便添加必需的条目。如果不更新构建路径，则 Java 编译器将不能解析对属于必需的插件类型的引用。

4. “运行时”页面

“运行时”页面显示有关运行时库的信息。当封装时，平台插件将提供它们在 JAR 库中的所有 Java 类。此页面用于定义源文件夹中的类在设计时是如何封装到库中的。在创建项目期间，向导已经创建了一个源文件夹和一个库。在此页面上，可以定义多个源文件夹和库，如图 14-31 所示。

缺省情况下，库具有“代码”类型，因此假定库包含可执行代码和资源。访问这种

库中的任何内容都会激活插件及其所有依赖项。如果插件中的库不用来包含可执行代码，则应将该库标记为其类型为“资源”。访问这种库中的“资源”不会激活插件，因此极大地提高了性能。

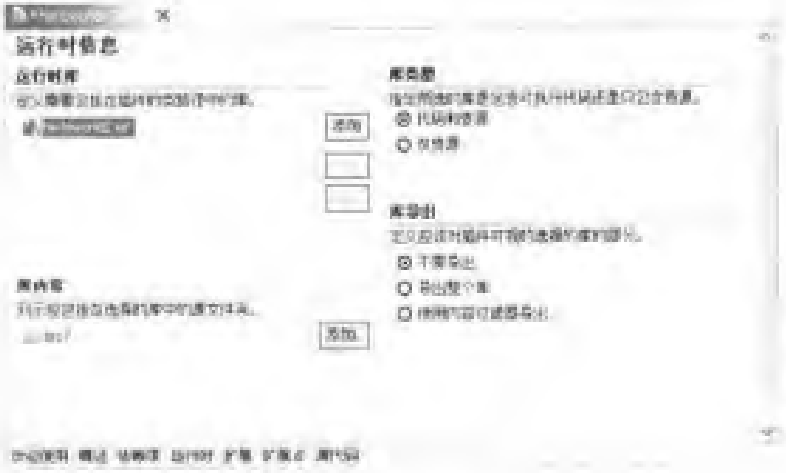


图 14-31　“运行时”页面

在本页中还可以确定库的导出规则。缺省情况下，其他人不能看到库中的任何类，如果不打算扩展插件，则“缺省”值是适当的。如果要使其他插件也可以看到用户的类，则必须导出库。可以导出整个库，或者只导出与提供的内容过滤器（例如，com.example.*）相匹配的那些部分。为一个库声明包前缀将使类装入速度提高 10%~15%。

5. “扩展”页面

“扩展”页面可用来浏览和编辑插件扩展。扩展是用来为平台添加行为的中央机制。除非插件是可供其他插件使用的简单的 Java API 库，否则，会将新行为作为扩展来添加。插件为扩展点（由所需要的插件来定义）添加行为。扩展语法必须与扩展点定义相匹配，才能正确地进行处理。合格的扩展点的总列表是由 PDE 通过遍历所有可视插件（工作区插件和外部插件）来计算的。由 PDE 提供的一般扩展向导仅扫描扩展点。如果提供了扩展模式，则向导将从模式中抽取语法和扩展点元数据。除了一般向导之外，还可以获得大量扩展模板。这些是可以用来生成插件项目的初始内容的相同模板。如果它们在此处存在，则允许用户创建扩展。可以使用简单视图来创建插件，稍后再添加编辑器或向导，如图 14-32 所示。

6. “扩展点”页面

“扩展点”页面可用来浏览和编辑由插件定义的扩展点。插件的扩展点可以供插件本身或者另一个插件使用。定义扩展点时，必须提供标识和名称。标识应该是一个简单名称。其他插件将使用全限定版本，它们会将插件标识用作前缀。例如，必须使用 com.example.xyz.foo 来引用扩展点 foo。除了显示扩展点的列表之外，此页面还显示一个图，它表示由其他插件添加到该扩展点的内容。用户将能够看到是谁真正扩展了自己的扩展点。

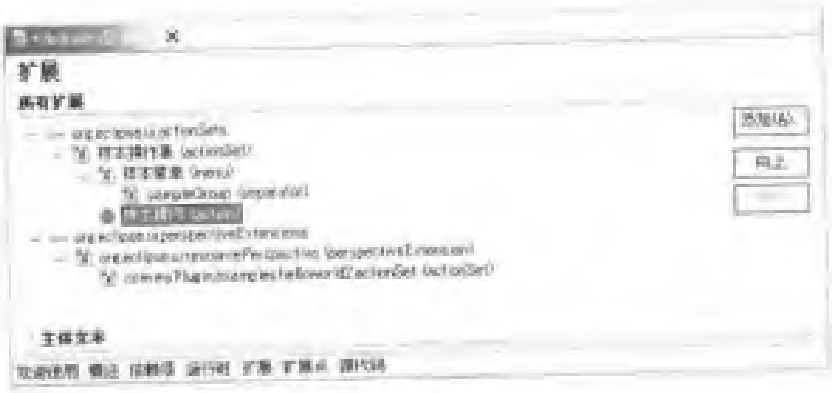


图 14-32 “扩展”页面

7. “源代码”页面

除了 GUI 页面之外，PDE 还提供了源代码页面“源代码”页面如图 14-33 所示。喜欢手工编辑 plugin.xml 文件的用户可以使用该页面。由于手工编辑该文件可能会导致在语法和语义上产生大量错误，因此，PDE 会在保存或切换选项卡时验证该文件，防止发生此类错误。



图 14-33 “源代码”页面

PDE 的另一个功能是：如果出现了语法错误，则 PDE 将锁定表单页面，直到修正了该错误为止。假如出现下面的语法错误，那么在保存文件的时候任务视图（如图 14-34 所示）会提示用户文件有错。



图 14-34　PDE 锁定表单页面

如果用户尝试着切换到任何不是源代码页的页面，则将看到如图 14-35 所示的对话框。



图 14-35　提示框

如果关闭存在错误的编辑器，然后再次打开它，则该编辑器会在“源代码”页面中打开，并且将不能访问其他页面，直到修正了源代码页的错误为止。

PDE 能够检测到的语法错误都显示在“窗口（W）”→“首选项（P）”中的“插件开发”→“编译器”页面的插件选项卡上，可以将每一项的级别设置为“错误”、“警告”或“忽略”，如图 14-36 所示。



图 14-36　编译器选项卡

14.6　运行插件

当在工作区中开发插件时，Java 编译器将编译 Java 源代码，并将*.class 文件放入 PDE 项目的 bin 目录中。当准备测试插件时，可以启动运行时工作台实例来测试新插件。JDT 插件提供了启动支持，PDE 通过定义自己的名为“运行时工作台”的启动器类型来使用启动支持。激活它的最简单方法是从菜单栏中选择“运行（R）”→“运行为（S）”→“运行时工作台”。这将立即创建另一个（运行时）工作台实例。

如果要配置启动“运行时工作台”的方式，则应从主菜单栏中选择“运行（R）”→“运行（N）”。显示的对话框如图 14-37 所示。



图 14-37　运行时工作台选项卡

可以创建几种配置，并且要为它们指定惟一的名称，每种配置具有不同的参数。有户可以通过在不同配置之间进行切换来在不同条件下测试插件。对于每种配置，可以定义主自变量，控制将要运行的插件和段，激活跟踪功能以及定义所有启动器公用的参数。

“工作区数据（D）”定义运行时工作台使用的工作区。PDE 将设置缺省值，但是可以更改它。可以选择将用来启动实例的 Java VM 以及 VM 和程序自变量。在任何时候，都可以使用“恢复缺省值（S）”按钮来恢复缺省值。

14.6.1　选择要运行的插件

在运行或调试时，可以改变 PDE 传递给运行时工作台的插件和段的列表。缺省情况下，配置“插件和段”选项卡来选择在“首选项”中启用的所有工作区插件和外部插件。但是，选择另一个单选按钮时将显示可以单独选择或取消选择的插件的列表，如图 14-38 所示。



图 14-38　选择要运行的插件

如果想要准确地知道正在传递给运行时工作台的所有插件来自何处，可按“插件路径（P）”按钮，打开一个窗口，它列示了所有插件的 URL。

为了支持跟踪而不需要编辑跟踪.options 文件，PDE 提供了“跟踪”首选项页面，如图 14-39 所示。该页面允许用户设置与装入相关的跟踪标志和名称过滤器。

要特别对待针对运行时类的跟踪，原因是它是最常见的，对于大多数插件开发者都

很有用。PDE 还提供了一般对话框，用来控制其他跟踪标志。“高级跟踪”首选项页面显示支持跟踪的所有插件。当选择了插件时，它的受支持跟踪选项就会显示在属性页中。可以对工作区插件和外部插件更改这些跟踪选项的值。



图 14-39 “跟踪”首选项页面

如果将跟踪支持添加到正在开发的插件中，则这些跟踪值将显示在此页面中。

可以单独选择工作区和外部插件，也可以作为一个组来选择。这使用户具有很大的灵活性，但是，同时又增加了产生无效配置的风险。当不能确定原始设置时，应小心使用此功能部件并按“回复（V）”按钮恢复缺省值。

14.6.2 对运行时进行跟踪

平台提供了一种机制，用于在运行时跟踪插件的活动，而不进行完全调试。它还允许用户使用跟踪标志，将跟踪信息打印在标准输出（或“控制台”视图）上。这些标志是在名为“.options”的文件中定义的，其语法如下。

```
<plug-in Id>/debug = true/false  (master switch)
<plug-in Id>/<tracing flag> = <value>
```

第一个行表示跟踪插件的主开关。如果此跟踪变量的值为 true 的话，调用插件类中的方法 isDebugging，并且将返回 true。其他跟踪标志由用户来定义，并且可使用以下方法来获取它们的值。

```
Platform.getDebugOption(optionName);
```

大多数平台插件都定义了跟踪标志，特别是平台核心。对于初级插件开发者来说，最感兴趣的一组跟踪标志是那些与类装入相关的跟踪标志，原因是它们允许跟踪插件在装入时产生的问题。

为了支持跟踪而不需要编辑跟踪.options 文件，PDE 在启动对话框中提供了“运行时跟踪”选项卡。所有跟踪设置都保存在启动配置中。这意味着可以使用不同的跟踪选项创建几种配置，并将它们互相作为备用。

为了允许其他开发者控制用户的插件的跟踪标志，需要让大家都知道这些选项。这通常是通过将.options 文件放入用户的插件中来实现的。该文件列出所有受支持的标志及其缺省值。PDE 将检测到此文件，并将它包含在“高级跟踪”首选项页面上的“工作区插件”的列表中。

为插件添加跟踪可遵循以下步骤：

（1）右击 com.myPlugin.examples.helloworld2 项目，选择“新建（W）”→“文件”，打开“新建文件”向导，建立一个名为“.options”的文件，如图 14-40 所示。



图 14-40　添加跟踪

（2）打开.options 文件，添加如下的内容，并保存：

```
com.myPlugin.examples.helloworld2/debug = true
com.myPlugin.examples.helloworld2/debug/flag = true
com.myPlugin.examples.helloworld2/debug/filter = *
```

（3）选择菜单栏上的“运行（R）”→“运行（N）”以打开启动对话框。打开“跟踪”菜单，选中“启用跟踪”选择项。现在，插件应该显示在“工作区插件”中。选择该插件时，它应该显示最新定义（就是在.options 文件中添加的内容）的标志及其缺省值，如图 14-41 所示。

这样，就完成了为插件添加跟踪。但是创建.options 文件只定义了可用性标志，允许其他插件开发者定义跟踪属性的值。用户仍然需要使用 Platform.getDebugOption()来检查插件代码中的跟踪属性的值。



图 14-41　显示新建的跟踪

14.6.3　在调试方式下运行

可以使用同一“运行时工作台”启动器来对插件进行调试，见图 14-42。除了标准的 Java 调试支持之外，没有特殊的 PDE 支持调试。调试插件的时候按调试工具栏按钮而不是按运行按钮。可以在 Java 代码中放置断点，一旦遇到这些断点时，就单步跳过源代码。PDE 确保工作区插件和外部插件在调试时不存在任何区别。用户可以从一个类跳到另一个类，而不需要知道正在查看的是与外部库相连接的只读源还是工作区中的可编辑源。如果目标平台与主机不相同并且目标平台是一个不包含源代码的产品，那么当控件移到平台类中时，用户不能够单步执行源代码。一些基于平台的产品可能会选择交付时不提供源代码，而将源代码作为产品 SDK 的一部分来提供。在这种情况下，要单独安装源代码。源代码是通过包含 ZIP 文件的插件交付的，并插入到 PDE 扩展点以注册源代码位置。当关联运行时库和源连接时，PDE 将扫描可用的源代码位置，以尝试查找相匹配的源 ZIP。

14.6.4　运行插件

从主菜单中选择“运行（R）”→“运行为（S）”→“运行时工作台”。另一个平台

实例会打开。如果在菜单栏上不能看到新的操作集，则需要通过在新的平台实例上选择“窗口（W）”→“定制透视图（Z）”→“其它”，并选择“样本操作集”（由用户添加的操作集）的复选框，单击“确定”从而激活新操作集。如图 14-43 所示。



图 14-42　运行时工作台选项卡

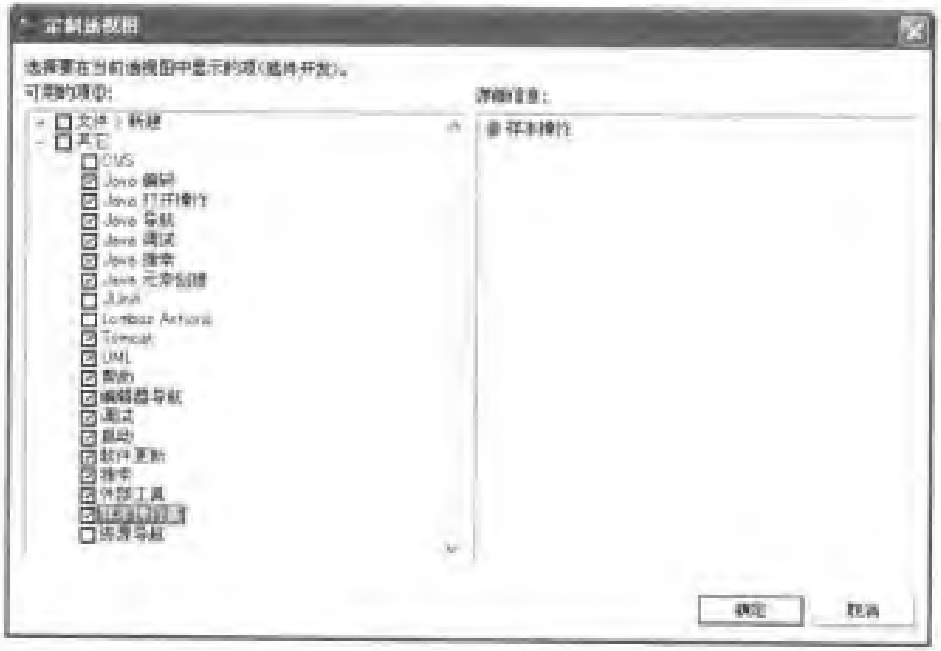


图 14-43　打开插件视图

“样本菜单”将会显示在菜单栏中，并且具有“样本操作”项，如图 14-44 所示。



图 14-44　样本菜单中具有样本操作项

选择它将会弹出信息为“Hello, Eclipse world”的提示框，如图 14-45 所示。



图 14-45　提示框

本章小节

在本章中讲解了 Eclipse 工作平台的体系结构，介绍了平台中的每个主要子系统本身是由一组插件构成的，这些插件实现一些重要功能并定义扩展点。通过制作一个简单的视图插件，读者了解了如何把自己的代码变成插件。

在本章中还讲解了什么是插件开发环境，怎样配置插件开发环境，以及“错误日志和插件注册表”这两个插件开发环境所提供的视图。通过一个详细的示例，讲解了插件开发环境是怎样为开发者“开发插件”而提供帮助的。

第 15 章　Eclipse 的界面 API

Eclipse 平台用户界面构建在工作台之上，它为用户提供了一个整体架构和一个可扩展的用户界面。用过 JBuilder 和 Eclipse 的编程人员会发现，在界面美观程度上 Eclipse 比 JBuilder 改善了很多，最新发布的 Eclipse 3.0 版的用户界面更加精致美观。这是因为 Eclipse 没有沿用 Java 的 AWT，Swing，而是自己开发了一套界面 API 包：SWT，JFace。

本章主要介绍以下内容：

- SWT 的介绍（包括 SWT 的特征、提供的组件、布局及 SWT 应用程序结构等）
- SWT 的使用（包括创建 SWT 项目及应用程序）
- SWT 线程问题
- 插件的用户界面框架 JFace

15.1　SWT 简介

标准小窗口工具箱 SWT 指的是由 Eclipse 项目提供的创建 GUI 所使用的工具。以大写“W”为开头字母的单词“Widget”，是指按钮、菜单、滚动条等用户接口（UI）的构成要素。

15.1.1　SWT 的特征

SWT 原来是由美国 IBM 公司开发的一种在 Eclipse 中使用的 IDE。伴随 Eclipse 作为一种开放性的源代码公布于世，IDE 也逐渐为世人所熟悉。

SWT 主要有以下特征：

1．动作迅速

SWT 经由 JNI 调用操作系统（以下简称 GS）的内部 API，因此动作迅速。Eclipse 自身的接口也是由 SWT 做成的。Eclipse 之所以能够被广泛地应用，原因之一就是 SWT 的动作迅速。

2．平台式的内部外观

由于在对用户的接口进行描述时调用了 OS 的内部 API，所以能够实现与平台式的内部应用程序相同的外观。

3．不是一种纯粹的 Java 程序

之所以说 SWT 不是一种纯粹的 Java 程序，其原因是 SWT 使用了 JNI。因此，在不

同的平台中，都需要有 SWT 所共有的资源库（Library）。也就是说目前广泛的 OS 中，包括 Windows，Linux，Solaris，QNX，AIX，HP-UX，MacOSX，Windows CE 等，只要拥有了 SWT 所使用的共有资源库，都可以使用 SWT。但是，需要读者注意的是，在编程过程中每个 Widget 都拥有各自的 dispose 方法，在清除 Widget 时，需要明确地调用 dispose 方法。

4. SWT 应用程序的代码不单独依存于某一平台

SWT 的应用程序是基于 Shell，Button 等 SWT 的类进行开发的。Shell，Button 通过带有内部 API 的 OS、Callback 以及 COM 调用共有的 DLL 向 OS 提出要求。由应用程序发出的调用内部 API 的要求全部被 Java VM，SWT 隐藏起来，因此，使用 SWT 所开发的应用程序不单纯依赖于某一平台。另外，虽然说基本的 Widget 不单纯依赖于某一个平台，但是在 SWT 中也可以使用平台所固有的一些技术。比如说，在 Windows 中可以将 IE 作为 ActiveX 的组件拷贝到画面中，类似的处理在 SWT 中也可以进行。

SWT 应用程序的平台处理过程示意图见图 15-1。

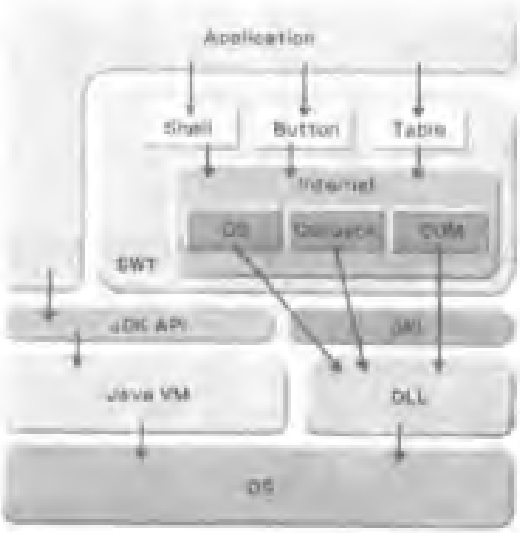


图 15-1 SWT 应用程序的平台处理过程

5. SWT 是 Eclipse 插件开发的必备工具

Eclipse 是基于 SWT 进行处理的，所以如果要开发 Eclipse 的插件，那么 SWT 是必备的开发工具。当然，在插件开发的过程中也可以使用 JFace，但是 JFace 本身也是利用 SWT 进行各种处理的工具，并且 JFace 不能够完全将 SWT 隐藏起来，所以掌握 SWT 的相关知识仍然很重要。

6. 灵活的许可文件

SWT 同 Eclipse 一样利用 CPL 进行发布。因此，不需要公布使用 SWT 做成的应用程序的源代码，因此也没有许可方面的限制（例如，不可商用使用的限制等）。

15.1.2　SWT 应用程序结构

典型的独立 SWT 应用程序具有下列结构：

- 创建显示，它表示 SWT 会话。
- 创建一个或多个 shell，它充当应用程序的主窗口。
- 创建 shell 内部所需的任何其他小窗口。
- 初始化小窗口的大小和其他必需的状态，为需要处理的小窗口事件注册侦听器。
- 打开 shell 窗口。
- 运行事件调度循环，直到发生退出情况为止（通常是在用户关闭主 shell 窗口的情况下）。
- 清除显示。

下面是一段简单的 SWT 应用程序的例子，该应用程序只是用来显示字符串“Hello World”，因此不需要向任何小窗口事件注册。

```
public static void main (String [] args) {
    //创建显示
    Display display = new Display ();
    //创建 一个或多个 shell
    Shell shell = new Shell (display);
    //创建 shell 内部的小窗口
    Label label = new Label (shell, SWT.CENTER);
    label.setText ("Hello_world");
    //始化小窗口的大小
    label.setBounds (shell.getClientArea ());
    //打开 shell 窗口
    shell.open ();
    //运行事件调度循环
    while (!shell.isDisposed ()) {
    if (!display.readAndDispatch ()) display.sleep ();
    }
    //清除显示
    display.dispose ();
}
```

15.1.3　SWT 提供的组件

在开发带有 GUI 应用程序的时候，有必要对所使用的工具具备什么样的组件进行了解。下面对 SWT 提供的组件进行详细介绍，并同时将 SWT 同 Java2 标准的 GUI 组件 Swing 进行比较。

1．显示（display）

显示表示 SWT 与底层平台的图形用户界面系统之间的连接。显示主要用来管理平台事件循环和控制用户界面线程与其他线程之间的通信。

对于大多数应用程序，可以遵循以上所使用的模式。在创建任何窗口之前必须先创建显示，并且当关闭 shell 时，必须清除显示。除非是设计多线程应用程序，否则，不需

要太多地考虑显示。

2. shell 组件

在 SWT 中把应用程序的结构窗口称为 shell。shell 是由 OS 平台窗口管理器管理的一个“窗口”。顶级 shell 用来显示子代创建的那些 shell，这些窗口是在用户使用应用程序时“移动、调整大小、最小化和最大化的”窗口。辅助 shell 是作为另一个 shell 的子代创建的那些 shell，这些窗口通常用作只存在于另一个窗口的上下文对话框窗口或其他瞬时窗口。

关于对 Shell 组件的说明见表 15-1。

表 15-1 对 Shell 组件的说明表

组 件	说 明	对应 Swing
Shell	拥有标题条、最小化/最大化/关闭按钮的窗口；此窗口中配置 widget	JFrame

3. 输入组件

这些组件（见表 15-2）表示 SWT 与底层平台的图形用户界面系统之间的连接。主要用来管理平台事件循环和控制用户界面线程与其他线程之间的通信，处理字符串的输入与输出，以及对应事件发生。

表 15-2 对输入组件的说明表

组 件	说 明	对应 Swing
Button	普通按钮，check 按钮，radio 按钮，toggle 按钮	JButton JCheckBox JRadioButton JToggleButton
Text	可以输入文字的按钮，可以指定单行、多行以及滚动行	JTextField JTextArea
Combo	Textbox 和下拉按钮相结合的 combobox，也可以禁止输入	JComboBox
Slider	能够进行数值调整，同其他的 Widget 同期滚动的 bar	JSlider

4.显示组件

显示组件（见表 15-3）用于显示文字、图片、一览等。

表 15-3 对显示组件的说明表

组 件	说 明	对应 Swing
Label	显示文字或图片	JLabel
Canvas	显示任意表格	JPanel
List	能够进行一览显示，可以从一览中指定任意一行或几行	JList
Table	显示二维表格，可以在列头指定列名	JTable
Tree	可以显示文件路径等带有亲子关系的要素	JTree
ProgressBar	显示进度条	JprogressBar

5. Menu/工具栏组件

该组件（见表 15-4）可以在窗口上配置 Menu 以及工具栏。

表 15-4　对 Menu/工具栏组件的说明表

组　件	说　　明	对应 Swing
Menu	在 menu bar 中显示[menu]、[help]等	JMenu
ToolBar	显示工具栏	JToolBar
CoolBar	显示可以使用鼠标托动的工具栏	无

6. 容器组件

容器组件（见表 15-5）还可以包括其他的 Widget。

表 15-5　对容器组件的说明表

组　件	说　　明	对应 Swing
Composite	包含其他 Widget	JPanel
Group	同 Composite 一样，可以包含其他 Widget。特别值得提出的是可以用线包围	JPanel + Border
TabFolder	通过 Tab 换页	JTabbedPane
SashForm	把 area　分为 .:。可以利用鼠标移动边界	JSplitPane

7. 对话框组件

SWT 可以使用如表 15-6 所示的六种对话框组件。

表 15-6　对“对话框组件”的说明表

组　件	说　　明	对应 Swing
MessageBox	显示 message	JOptionPane
FileDialog	指定文件	JFileChooser
DirectoryDialog	指定路径	JFileChooser
ColorDialog	指定颜色	JFileChooser
FontDialog	指定字体	无
PrintDialog	指定打印机	Java Print API 的 PrinterJob

8. 布局管理组件

该类组件（见表 15-7）用于指定格式，调整容器组件窗口内的配置。

表 15-7　布局管理组件

组　件	说　　明	对应 Swing
FillLayout	填补空隙，水平或者垂直配置 Widget	BoxLayout
FormLayout	参照其他窗口进行布局	SpringLayout
RowLayout	水平或者垂直配置窗口，可以指定 margin 或者 Widget 之间的空间	BoxLayout FlowLayout

（续表）

组　件	说　　明	对应 Swing
GridLayout	可以进行类似于 HTML 的 table 标签的配置，可以指定 owspan 或 colspan	GridBagLayout
StackLayout	只显示最上层的 Widget	CardLayout

9. 事件调用组件

该类组件（见表 15-8）启动键值输入或 Widget 选择等各种各样事件的接口。通过在 Widget 中登录装有 listener 的类，实现事件发生时应用程序的逻辑调用。

表 15-8　对事件调用组件的说明表

组　件	说　　明	对应 Swing
SelectionListener	选择 Widget 时被启动，在大多数 Widget 中都有这样的设定	ActionListener
KeyListener	由键盘输入时被启动	KeyListener
FocusListener	Widget 得到或者失去焦点的时候被启动	FocusListener
MouseListener	在 Widget 上按动鼠标，移动鼠标或者双击鼠标时被启动	MouseListener

10. 其他组件

有关其他组件的说明见表 15-9。

表 15-9　对其他组件的说明表

组　件	说　　明	对应 Swing
Display	显示 SWT 间平台之间的连接，在应用程序的开头创建，结束删除	无
Program	与文件的扩展名相关联的外部程序	无
SWT	生成 Widget 时，将指定的类型定义为定量	无

15.1.4　布局

SWT 定义了布局，它提供组合体中子代小窗口的通用定位和缩放。布局是抽象类布局的子类，可以在 org.eclipse.swt.layout 包中找到 SWT 标准布局。

当对小窗口重新调整大小和定位时，需要了解下面一些常规定义。

- 位置：指小窗口在父代小窗口中的 x 和 y 的坐标位置。
- 首选尺寸：指显示小窗口内容所需要的最小尺寸。对于每种小窗口，其计算方法是不同的。在组合体的情况下，首选尺寸是包含采用首选尺寸的组合体及其所有子代的最小尺寸。
- 客户区域：指小窗口的内容区域的大小。
- 修剪：指小窗口的客户区域与它的实际边界之间的距离。修剪被小窗口的边框或边框边缘的额外空间占用。修剪的大小和外观与小窗口和平台有关。
- 间距：指某些布局支持布局中的各个小窗口之间存在间距。
- 页边距：指某些布局支持布局的边缘与和边缘相邻的小窗口之间存在页边距。

这些概念与应用程序有关，布局是封装调整大小功能以便重复使用的便捷方法。从下面两段代码片段可以看出使用布局的快捷方便。第一段应用程序的目的是使用调整大小回调来将标号的大小调整为它的父代 shell 的大小：

```
Display display = new Display ();
Shell shell = new Shell (display);
Label label = new Label (shell, SWT.CENTER);
shell.addControlListener(new ControlAdapter() {
   public void controlResized(ControlEvent e) {
      label.setBounds (shell.getClientArea ());
   }
});
```

以下代码片段使用布局来获得相同的效果：

```
Display display = new Display ();
Shell shell = new Shell (display);
Label label = new Label (shell, SWT.CENTER);
shell.setLayout(new FillLayout());
```

可以看出对于此简单示例，使用布局减少了应用程序代码，对于更复杂的布局，简化则更大。SWT 提供了可用于许多情况的四个缺省布局类，下面分别介绍各种布局。

1．填充布局（FillLayout）

FillLayout 是最简单的布局类。它将小窗口布局在单行或单列中，强制它们全部使用相同的大小。最初，所有小窗口都与最高的小窗口一样高，并且与最宽的小窗口一样宽。FillLayout 不会覆盖，并且不能指定页边距和间距。

可以使用 FillLayout 对任务栏或工具栏中的按钮进行布局，或者将复选框叠放在组中。当组合体只有一个子代时，也可以使用 FillLayout。在以上示例中，FillLayout 强制标号完全填充它的父代 shell。

表 15-10 总结了填充布局的属性。

表 15-10　填充布局的属性表

属　性	描　　述
调　整	可以是 HORIZONTAL（行布局）或 VERTICAL（列布局）之一

2．行布局（RowLayout）

RowLayout 在行中布局小窗口，类型字段用于控制小窗口是在水平行中放置，还是在垂直列中放置。它能可选择地覆盖小窗口，根据需要创建足够多的行或列来显示它们。它还提供布局的每个边缘上可配置的页边距，以及布局中的小窗口之间可配置的间距。如果调整 RowLayout，则组合体中保留的额外空间将被分配为小窗口之间的页边距。

用户还可以在 RowData 对象（应该在小窗口中使用 setLayoutData（Object）来设置它）中指定其中的每个小窗口的高度和宽度。

表 15-11 总结了行布局的属性。

表 15-11 行布局的属性表

属 性	描 述
调 整	可以是 HORIZONTAL（行布局）或 VERTICAL（列布局）之一
底空白	指定要沿布局底边放置的像素数
左空白	指定要沿布局左边放置的像素数
右空白	指定要沿布局右边放置的像素数
顶空白	指定要沿布局顶部放置的像素数
包 装	指定是否应强制所有小窗口具有相同大小
间 隔	指定一个单元与其相邻单元之间的像素数
类 型	可以是 HORIZONTAL（行布局）或 VERTICAL（列布局）之一
回 绕	指定若在当前行或列中没有足够空间，是否将控件回绕到下一行或列

表 15-12 总结了行数据的属性。

表 15-12 行数据的属性表

属 性	描 述
宽 度	指定以像素为单位的单元宽度
高 度	指定以像素为单位的单元高度

3. 表单布局（FormLayout）

FormLayout 使用附件模型来布局小窗口。可以为小窗口的每侧定义一个表单附件（FormAttachment），并将其存储在它的布局数据（FormData）中。FormData 可以用来指定小窗口请求的宽度和高度，这些值不能与附件约束发生冲突。

FormAttachment 用于将指定的小窗口一侧连接到父代组合体或布局中的另一个小窗口。通常不设置小窗口所有侧面上的附件，普遍的做法是仅指定一个水平（左或右）附件和一个垂直（顶部或底部）附件，并允许小窗口采用在它们的 FormData 中指定的大小或其首选尺寸（如果在 FormData 中未指定大小的话）。

表 15-13 总结了表单布局的属性。

表 15-13 表单布局的属性表

属 性	描 述
marginHeight	指定将对布局顶和底边设置的垂直页边距的像素数目
marginWidth	指定将对布局左和右边设置的水平页边距的像素数目

表 15-4 总结了表单数据的属性。

4. 网格布局（GridLayout）

GridLayout 是功能强大且更加复杂的布局之一。GridLayout 在网格中布局小窗口，

会有许多可配置的参数。当调整组合体的大小时，用来控制网格行和列的缩放行为。

表 15-14　表单数据的属性表

属　性	描　　述
顶　部	指定控件顶部的附件
左　边	指定控件左边的附件
底　部	指定控件底部的附件
右　边	指定控件右边的附件
宽　度	指定表单中控件的首选宽度（以像素为单位）
高　度	指定表单中控件的首选高度（以像素为单位）

表 15-15 总结了表单附件的属性。

表 15-15　表单附件的属性表

属　性	描　　述
对　齐	指定与控件连接的控件端对齐。DEFAULT 指示应将小窗口连接到指定控件的相邻一侧。对于顶部和底部附件，使用 TOP、BOTTOM 和 CENTER 来指示小窗口的指定一侧连接到控件的指定一侧。对于左边和右边的附件，使用 LEFT、RIGHT 和 CENTER 指示小窗口的指定一侧连接到控件的指定一侧（例如，使用 TOP 指示附件小窗口的顶部应连接到指定的控件的顶部。）
控　件	指定附件小窗口连接至的目标控件
分　母	指定定义附件的等式 y=ax+b 中“a”项的分母
分　子	指定定义附件的等式 y=ax+b 中“a”项的分子
偏移量	指定控件侧从附件位置的偏移量（以像素为单位）。可以是正或负。这是定义附件的等式 y=ax+b 中的“b”项

GridLayout 定义用来控制布局的总体战略的 API。最重要的属性为 numColumns，它确定网格的水平大小。通常，当首次设计窗口的外观时，就要决定此值。网格中的小窗口的次序与创建它们的次序相同。要更改网格中小窗口的次序，可以使用控件方法 moveAbove（Control）和 moveBelow（Control）。这些方法允许小窗口在布局中互相之间前后插入。

表 15-16 总结了网格布局的可配置参数。

表 15-16　网格布局的可配置参数表

属　性	描　　述
horizontalSpacing	一个单元的右边缘与它相邻单元的左边缘之间的像素数目
makeColumnsEqualWidth	指定是否应该强制所有列为相同宽度
marginWidth	用于网格的右边缘和左边缘上的页边距的像素数目

（续表）

属　性	描　　述
marginHeight	用于网格顶部和底部边缘上的页边距的像素数目
numColumns	用来生成网格的列数
verticalSpacing	个单元格底部边缘与它相邻单元的顶部边缘之间的像素数目

网格布局支持网格中的每个小窗口的许多其他布局参数。这些属性是在网格数据（GridData）对象中指定的，必须将 GridData 设置为网格中的每个小窗口的布局数据。

GridData 类定义样式常量，它允许在 GridData 构造器中指定布局参数的常用组合。还可以使用 GridData 中的公共方法来单独设置这些属性。通过使用 GridData 可以获得一些高动态和复杂的布局，将不同小窗口的这些值组合在网格中可以生成许多不同的组合的布局，并且允许小窗口跨越单元的能力可以做成许多甚至不像网格的布局。

表 15-17 总结了网格数据的可配置参数。

表 15-17　网格数据的可配置参数表

属　性	描　　述
grabExcessHorizontalSpace	指定单元是否应该增大，以便使用网格中可用的任何额外水平空间。根据小窗口和它们的网格数据计算网格中每单元大小之后，将把"组合体"中余下的任何额外空间分配给需要额外空间的那些单元
grabExcessVerticalSpace	指定单元是否应该增大，以便使用网格中可用的任何额外的垂直空间
heightHint	指定小窗口的最低高度（以及包含小窗口的行的最低高度）
horizontalAlignment	可以是 BEGINNING、CENTER、END 和 FILL 的其中之一。FILL 意味着将调整小窗口的大小，以便占据它的网格单元的整个宽度
horizontalIndent	小窗口与它的网格单元的左边缘之间的像素数目
horizontalSpan	指定小窗口应该跨越的网格中的列数。缺省情况下，小窗口占据网格中的一个单元，它可以通过增大此值来水平获取其他单元
verticalAlignment	可以是 BEGINNING、CENTER、END 和 FILL 的其中之一。FILL 意味着将调整小窗口的大小，以便占据它的网格单元的整个高度
verticalSpan	指定小窗口应该跨越的网格中的行数。缺省情况下，小窗口占据网格中的一个单元，它可以通过增大此值来垂直获取其他单元
widthHint	指定小窗口的最小宽度（以及包含小窗口的列的最小宽度）

5. 堆栈布局（StackLayout）

StackLayout 创建小窗口的概念上的叠放。所有小窗口都设置为具有相同的大小和位置，并显示指定为位于顶部的一个小窗口。可通过将布局的 topControl 字段设置为期望的小窗口，然后在具有叠放布局的组合体上调用 layout()来更改指定的顶部小窗口。可相应地通过设置布局的 marginHeight 和 marginWidth 属性来指定布局使用的页边距的高度和宽度。

表 15-18 总结了堆栈布局的属性。

表 15-18　堆栈布局的属性表

属　性	描　　述
marginHeight	指定要沿着布局的顶边和底边放置的像素数
marginWidth	指定要沿着布局左边和右边放置的像素数
topControl	指定叠放中概念上的顶部控件，将不显示所有其他控件

15.1.5　SWT 生命周期

当应用程序创建小窗口时，SWT 将立即创建底层平台小窗口。因此，不需要根据底层 OS 小窗口是否存在而执行不同操作的代码。它还允许将小窗口的大部分数据保存在平台层而不是在工具箱中复制，这就意味着工具箱的小窗口生命周期的概念必须符合底层图形用户界面系统的规则。

1．小窗口创建

创建小窗口时，大多数图形用户界面平台都要求指定父代。由于只要创建了工具箱小窗口，SWT 就会创建平台小窗口，因此，必须在构造函数中为小窗口指定父代小窗口。

2．样式位

在创建小窗口时，某些小窗口属性必须在 OS 中设置，并且不能进行后续更改。例如，列表可以是单个或多个选择，可以具有滚动条也可以没有滚动条。

必须在构造函数中设置这些属性（称为样式）。所有小窗口构造函数都采用 int 自变量，该自变量指定所有期望样式的位宽。在某些情况下，特定样式是一种提示，这意味着它可能在所有平台上都不可用，但是在不支持它的平台上将适当地忽略它。

样式常量作为公共静态字段存在于 SWT 类中，SWT 的“API 参考”中包含了每个小窗口类的可应用常量的列表。

3．资源处理

在 SWT 下面的 OS 平台需要显式地分配和释放 OS 资源。为了配合小窗口工具箱中的平台应用程序结构的 SWT 设计原理，SWT 要求显式地释放已经分配的任何 OS 资源。在 SWT 中，Widget.dispose()方法用来释放与特定工具箱对象相关联的资源。

经验法则是，如果创建对象，则必须清除该对象。以下进一步说明此原理的特定规则。

- 当使用构造函数来创建小窗口或图形对象时，使用完时必须手工来清除该对象。
- 如果没有使用构造函数就获取小窗口或图形对象，则一定不能用手工来清除该对象因为没有分配该对象。
- 如果将对小窗口或图形对象的引用传送至另一个对象，则一定要小心，仍在使用该对象时一定不要清除该对象。
- 当用户关闭 shell 时，将递归地销毁 shell 及其所有子代小窗口。在此情况下，不

需要清除小窗口本身，但是要释放与那些小窗口一起分配的所有图形资源。

- 如果为了在其中一个小窗口的生命周期内使用而创建图形对象，则在清除小窗口时必须清除图形对象，这可以通过向小窗口注册销毁侦听器，并在接收到销毁事件时释放图形对象来实现。

这些规则有一个例外：简单的数据对象（例如，矩形和点）不使用操作系统资源，它们没有 dispose()方法，因此也不需要释放它们。

15.2 SWT 应用程序开发

在使用 SWT 时，首先要获得资源库。如果是使用 Eclipse 进行开发的话，那么直接使用 Eclipse 本身带有的资源库就可以了。这里所使用的 Eclipse 版本为 2.1.2，Eclipse 的安装路径为 C:/eclipse。如果读者使用的是 Eclipse 升级后的版本，那么插件包名等可能会发生变化，请留意。最新发布的Eclipse 3.0版本的SWT插件包名为org.eclipse.swt.win32_3.0.0。

在使用 SWT 中，下面两个资源库是必需的。

- SWT 的 Java 资源库

SWT 类必须具备的.jar 资源库：C:\eclipse\plugins\org.eclipse.swt.win32_2.1.2\ ws\win32\ swt.jar

- 每种平台的通用资源库

Windows 下必须具备的通用资源库：C:\eclipse\plugins\org.eclipse.swt.win32_2.1.2\ os\win32\x86\swt-win32-2135.dll

在部署应用程序时，可以通过设定应用程序的类文件、SWT 的 Java 资源库、各种平台的通用资源库来实现程序的部署。

即使没有使用 Eclipse 的话，也可以在 Eclipse 的下载页面中得到 SWT 单独的资源库。下面是 Eclipse 的下载网址 http://download2.eclipse.org/downloads/index.php，单击 Latest Release 中的“2.1.2”，就可以在 SWT Binary and Source 中下载相应的文件。

15.2.1 创建 SWT 项目

在 Eclipse 上创建 SWT 的项目并不是一件难事，下面就来介绍如何创建 SWT 项目及设定必要的 Library。

1. 新建项目并设定库文件

通过“文件（F）”→“新建（N）”→“项目（R）”，打开“新建项目”的对话框，如图 15-2 所示。选择“Java”→“Java 项目”。

单击“下一步（N）”，在接下来显示的画面（见图15-3）中输入项目名。例如，本次输入的项目名是 swtPrj。



图 15-2　新建工程对话框



图 15-3　新建 Java 项目对话框

单击“下一步（N）”，在图 15-4 的画面中设定构建路径上的源文件夹和输出文件夹。本次输入如下内容：

构建路径上的源文件夹（H）：swtPrj/src

缺省输出文件夹（T）：swtPrj/bin

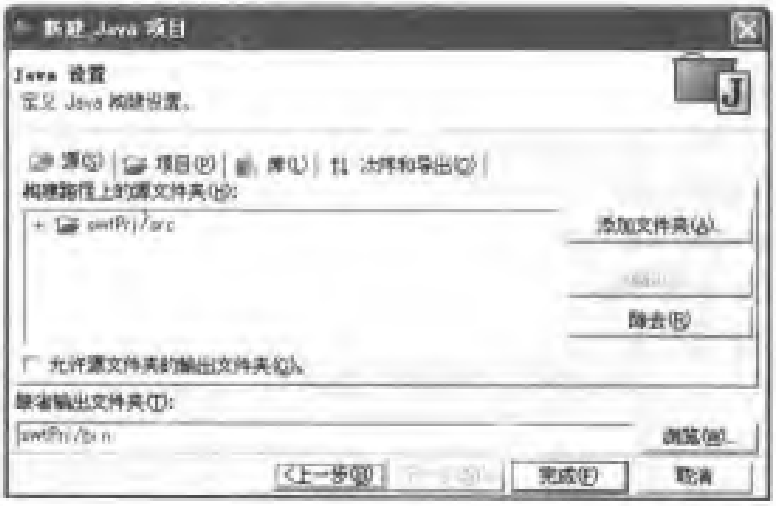


图 15-4　定义 Java 构建设置

选择“库（L）”选项卡，指定 SWT 的 Java 资源库。单击“添加外部 JAR（X）”的按钮（见图 15-5），在显示出来的“JAR 选择”的对话框中，选择 SWT 的 Java 资源库，

即：C:\eclipse\plugins\ org.eclipse.swt.win32_2.1.2\ ws\ win32\swt.jar。



图 15-5 添加库文件 swt.jar

单击“完成（F）”后就作成了一个 SWT 项目。

为了启动项目，还要准备平台所需的通用资源库，这些通用资源库在使用 SWT 进行编码的过程中暂时使用不到，但是在启动或者查错的过程中是必需的。将前面所提到的通用资源库拷贝到项目的路径下。

把 C:\eclipse\plugins\org.eclipse.swt.win32_2.1.2\os\win32\x86\swt-win32-2135.dll 拷贝到 C:\eclipse\workspace\swtPrj 中。如果不想进行该项操作的话，请在应用程序启动时，指定 [-Djava.library.path=C:\eclipse\plugins\org.eclipse.swt.win32_2.1.2\os\win32\x86] 为 Java VM 的参数 option，然后再执行程序。

2. SWT 源程序的设定

如果将 SWT 的源程序同 SWT 的 Java 资源库联合起来，在开发过程 SWT 类的 Javadoc 注释就会自动弹出，非常方便。

以右击“包资源管理器”上的 swt.jar，打开“属性”对话窗口（见图 15-6）。选择“Java 源代码连接”→“外部文件”，进行如下指定：

```
C:\eclipse\plugins\org.eclipse.platform.win32.source_2.1.2\src\org.eclipse.s
wt.win32_2.1.2\ws\win32\swtsrc.zip
```

如果下载的是一个 SWT 单体的话，那么在下载所得到的部件中就已经存在了 swtsrc.zip 文件，因此，只需指定该文件就可以了。

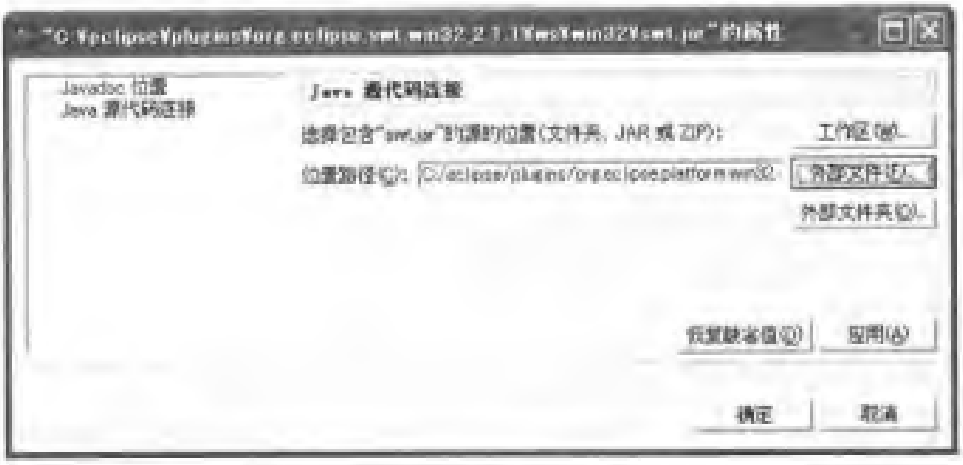


图 15-6　Java 源代码连接

15.2.2　创建 SWT 应用程序

本节通过编辑一个简单的 SWT 应用程序，对 SWT 进行说明，并且详细介绍利用 Eclipse 做成 SWT 应用程序的基本方法。

下面要做的 SWT 应用程序是为了显示 CSV 文件的 CSV 视图。CSV 视图的整体外观如图 15-7 所示。



图 15-7　CSV 视图

该应用程序的要求如下：

- 能够完成由菜单开始，经过读入文件选择，终止于应用程序完了的过程。
- 将读入文件的文件名显示在 text box 中。
- 通过表，显示 CSV 文件的内容。列名为[列 1]、[列 2]等，仅显示 CSV 文件的最大列数。
- 设置能够显示目前状况的状态条，包括：
 - 当选择表内的某行时，显示行号；

■ 显示读入文件所花费的时间。

按照下面的顺序制作该程序。

- 做成 Shell(Window)；
- 配置指向 Shell 的 Widget；
- 读入 CSV 文件，显示处理的实现；
- 面向 Widget 的 Event Listener 的登录。

1．创建 Shell

首先要完成能够显示窗口的工作。

通过 Eclipse 菜单上的“文件（F）”→“新建（N）”→“类（C）”，打开新建 Java 类的对话框，并输入如下信息。

源文件夹（D）：swtPrj/src

包（K）：csvviewer

名称（M）：CSVViewer

最后，按“完成（F）”按钮。

对 Eclipse 生成的 CSVViewer.java 类进行如下编辑。

例 15-1：CSVViewer.java

```
package csvviewer;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
/**
 * CSVViewer
 */
public class CSVViewer {
    private Shell shell;
    private Display display;
    public Shell open(Display display) {
        this.display = display;
        //生成一个或一个以上的 Shell，作为程序所需的主窗口
        shell = new Shell(display);    // (2)
        shell.setText("CSVViewer");
        //打开 Shell
        shell.open();        // (6)
        return shell;
    }
    public static void main(String[ ] args) {
        //创建显示
        Display display = new Display();  // (1)
        CSVViewer viewer = new CSVViewer();
        Shell shell = viewer.open(display);
        // 运行事件调度循环
        while (!shell.isDisposed()) {    // (7)
            if (!display.readAndDispatch())
            display.sleep();
        }
        //清除 Display
        display.dispose();      // (8)
    }
}
```

一般情况下，SWT 程序的处理流程为（顺序号同编码中的序号对应）：

① 创建显示。

② 生成一个或一个以上的 Shell，作为程序所需的主窗口。

③ 生成窗口内所需的其他 Widget。

④ 初始化 Widget 的大小以及其他必要的状态。

⑤ 向执行处理所必要的 Widget 中登录 Listener。

⑥ 打开 Shell。

⑦ 通过 Display 的 readAndDispatch 方法和 sleep 方法循环主窗口中的事件调度，直到主窗口被关闭为止。

⑧ 直到主窗口被关闭，才能清除 Display。

到此为止，已经在 CSVViewer 中写入了上面的①、②、⑥、⑦、⑧的步骤，请参照上面的编码中标注的顺序号。

在 CSVViewer 的源代码被选择的状态下，选择 Eclipse 菜单的“运行（R）”→“运行为（S）”→“Java 应用程序”，运行程序。正确的结果会出现图 15-8 所示的窗口。



图 15-8　窗口作成后的正确结果

这样的结果还不能称之为真正的应用程序，所以先关闭这个窗口。

2. Shell 的 Widget（GUI 组件）的配置

接下来进行 SWT 程序的处理流程中的步骤③，生成窗口内所需的其他 Widget 和步骤④初始化 Widget 的大小以及其他必要状态的处理。

下面按照每一项需要修改的项目，向读者进行说明。对原程序中变更及增加的内容进行了斜体表示。

（1）实例变量的追加。首先需要事先对其他方法中将要使用到的 Widget 定义为实例变量。

```
private Shell shell;
private Display display;
private Text textBox;
private Table table;
private Label statusBar;
private MenuItem mItemOpen;
```

（2）指定网格布局（GridLayout）。在 CSVViewer 的 open 方法内，在生成 Widget 之前，指定 shell 为 GridLayout。新增代码行如下：

```
shell = new Shell(display);    //(2)
shell.setText("CSVViewer");
shell.setLayout(new GridLayout(1, true));
```

由于指定 GridLayout 的结构参数 numColumns 为 1，所以向 Shell 中追加的 Widget 就排列竖向的一列。

（3）生成组合体（Composite）。进行完 Shell 设定之后，为了能够生成 textbox 和 table，需要生成组合体。在组合体中也应该像 Shell 一样，指定为 GridLayout 布局。

```
//生成组合体
Composite comp = new Composite(shell, SWT.NO_FOCUS);
comp.setLayout(new GridLayout(1, true));
GridData compGrid = new GridData();
compGrid.horizontalAlignment = GridData.FILL;
compGrid.verticalAlignment = GridData.FILL;
compGrid.grabExcessHorizontalSpace = true;
    compGrid.grabExcessVerticalSpace = true;
comp.setLayoutData(compGrid);
```

另外，设定 GridData，对作为父对象的 shell 的 GridLayout，进行能够将整个幅变宽的设置。

（4）生成文本框（textbox）。生成组合体之后，需要生成单一行的 textbox。并在父对象中指定刚才做成的 Composite。

```
    // 生成一行textbox
textBox = new Text(comp, SWT.SINGLE | SWT.BORDER);
textBox.setText("");
GridData textGrid = new GridData();
textGrid.horizontalAlignment = GridData.FILL;
textGrid.grabExcessHorizontalSpace = true;
textBox.setLayoutData(textGrid);
```

在生成时，是单一行还是复数行由结构的 styleflag 指定。这次指定 BORDER，目的是为了能够使生成的 textbox 带边框，可以通过演算子“|”指定多个 flag。

另外，设定 GridData，对作为父对象的组合体的 GridLayout，进行能够将整个幅变宽的设置。

（5）生成表（table）。接下来，生成显示 CSV 文件数据的表。在父对象中指定 Composite。同时指定了 MULTI，FULL_SELETCTION，BORDER 这三个 flag。MULTI，处理复数列的问题。FULL_SELETCTION，处理单行全选的问题。BORDER，处理 Table 带边框的问题。

```
    // 表的生成
 table=new Table(comp,SWT.MULTI | SWT.FULL_SELECTION | SWT.BORDER);
GridData tableGrid = new GridData();
 tableGrid.horizontalAlignment = GridData.FILL;
 tableGrid.verticalAlignment = GridData.FILL;
 tableGrid.grabExcessHorizontalSpace = true;
 tableGrid.grabExcessVerticalSpace = true;
```

```
table.setLayoutData(tableGrid);
    //线的可视表示
table.setLinesVisible(true);
    //题头可视
table.setHeaderVisible(true);
    // 列的题头可视
TableColumn col = new TableColumn(table, SWT.LEFT);
col.setText("列1");
col.setWidth(50);
```

设定 GridData，对作为父对象 的 Composite 的 GridLayout，进行能够将整个幅变宽，高度变高的设置。

通过调用 setLinesVisible 和 setHeaderVisible 方法，达到显示线条和项目头的目的。

另外，这里需要显示的是 CSV 文件未读入状态的表，所以只分了一列。

（6）生成状态栏（statusBar）。因为在 SWT 中不存在所谓的状态栏的项目，所以需要设定一个显示文字的标签。在父对象中指定 Shell。

```
//statusBar 的生成
statusBar = new Label(shell, SWT.LEFT);
GridData gridData = new GridData();
gridData.horizontalAlignment = GridData.FILL;
statusBar.setLayoutData(gridData);
statusBar.setText("指定 CSV 文件");
```

另外，设定 GridData，对作为父对象 的 Composite 的 GridLayout，进行能够将整个幅变宽的设置。

（7）生成菜单（Menu）。在 Shell 中设定菜单栏（MenuBar），菜单栏作为 Shell 的子要素同 Shell 构成以下关系的菜单。

```
菜单栏 MenuBar+
          +菜单项 MenuItem(“文件”)+
                        +菜单 Menu+
                                  +MenuItem(“打开”)]
                                  +MenuItem(“关闭”)
```

```
    //菜单生成
Menu menubar = new Menu(shell, SWT.BAR);
shell.setMenuBar(menubar);
MenuItem mItemFile = new MenuItem(menubar, SWT.CASCADE);
mItemFile.setText("文件(&F) ");
Menu menuFile = new Menu(mItemFile);
mItemFile.setMenu(menuFile);
mItemOpen = new MenuItem(menuFile, SWT.PUSH);
mItemOpen.setText("打开(&O) ");
new MenuItem(menuFile, SWT.SEPARATOR);
MenuItem mItemExit = new MenuItem(menuFile, SWT.PUSH);
mItemExit.setText("关闭(&X) ");
```

菜单项（“文件”），作为子要素指定 CASCADE（层叠）。菜单项（“打开”）和菜单项（“关闭”），指定 PUSH，以便在选择时构成事件。

完成上述程序编码后，即可运行该代码，并会出现图 15-9 所示的窗口。



图 15-9　窗口作成结果

一些必要的 GUI 组件都已经完成了，可以变更一下窗口的大小，然后确认一下 GridData 是否按照式样要求进行配置。但是，到目前为止的 CSVViewer，按动其中的“打开”、“关闭”按钮并没有什么反应。继续进行下面的工作，就是“向执行处理所必要的 GUI 组件中登录 Listener”，追加选择 Menu 时的一些处理。

3. 进行 CSV 文件读入和显示的处理

首先从处理的实例化开始。

（1）追加实例变量。在代码的开头追加如下的实例变量：

```
// 文件转换成*.csv格式
private final String[] EXTENSIONS = { "*.csv", "*" };
// 读入的文件名
private String fileName;
// 读入文件的测试时间
private long beginTime;
```

在上面的编码中，定义了下面三个变量。

- FileName：读入 CSV 文件名的变量。
- EXTENSIONS：使用文件下载功能时的文件扩展名。
- BeginTime：测定读入时间的变量。

（2）将表格生成作为一个方法抽取出来。在调用 Menu 时，需要有重新生成表格的处理，所以应该把生成表格的部分作为一个方法抽取出来，这样可以避免编码重复。

拖动 open 方法内需要抽出的部分，选择 Eclipse 菜单的“重构”→“抽出方法”，方法名为 *createTable*，然后单击“确定”。

```
public Shell open(Display display) {
......

//表的生成
createTable(comp);

......
shell.open();
 return shell;
 }

private void createTable(Composite parent) {
```

```
        table = new Table(comp,
          SWT.MULTI | SWT.FULL_SELECTION | SWT.BORDER);
        GridData tableGrid = new GridData();
       tableGrid.horizontalAlignment = GridData.FILL;
        tableGrid.verticalAlignment = GridData.FILL;
        tableGrid.grabExcessHorizontalSpace = true;
        tableGrid.grabExcessVerticalSpace = true;
        table.setLayoutData(tableGrid);
        // 线的可视表示
       table.setLinesVisible(true);
        // 题头可视
table.setHeaderVisible(true);
        // 列题头的设定
       TableColumn col = new TableColumn(table, SWT.LEFT);
        col.setText("列 1");
       col.setWidth(50);
        }
```

（3）取得文件名的处理。接下来需要编写代码，主要目的是在打开文件窗口时，取出文件名。

```
private void getOpenFileName() {
FileDialog openDialog = new FileDialog(shell, SWT.OPEN);
 openDialog.setFilterExtensions(EXTENSIONS);
 filename = openDialog.open();
}
```

（4）进行 CSV 文件读取前的处理。取到文件名之后，需要进行文件读取前的相关处理。

```
private void loadBegin() {
   // [打开] 菜单选择不可
mItemOpen.setEnabled(false);

// 设定文件框名
textBox.setText(fileName);

 // 列数变化后表再次生成
Composite comp = table.getParent();
table.dispose();
createTable(comp);
 comp.layout();

// 状态栏的表示
statusBar.setText("读入中...");

//读入测试时间
beginTime = System.currentTimeMillis();
}
```

在读取文件的过程中，不允许读取新的文件，也就是说 Menu 的“打开”按钮不可选。在 SWT 的 Table 中，一旦设定过的字段是不能够被清除的。因此，需要重新再生成表。在重新生成表时，需要调用刚才抽取出的 createTable 方法。

改变文本框和状态条的显示内容，为了能够达到测试读入时间的目的，需要用 System.currentTimeMillis 方法取出现在的时间。

（5）进行读取 CSV 文件的处理。当文件存在时，读入逗点隔开的数据，并且设置

到表中。具体程序代码如下：

```
private void loadFile() {
    BufferedReader reader = null;
try {
    File file = new File(fileName);
    if (!file.exists()) {
      return;
     }
// 文件打开
reader = new BufferedReader(new FileReader(file));
 // 循环处理每一行
 String line = null;
 while ((line = reader.readLine()) != null) {
   String[ ] datas = line.split(",");
  for (int I = table.getColumnCount(); i < datas.length; i++) {
    TableColumn column = new TableColumn(table, SWT.LEFT);
   column.setText("列" + (I + 1));
   column.setWidth(50);
}
// 行的设定
TableItem item = new TableItem(table, SWT.NULL);
item.setText(datas);
 }
} catch (IOException e) {
  e.printStackTrace();
} finally {
  if (reader != null) {
try {
  reader.close();
  } catch (IOException e) {
          e.printStackTrace();
     }
    }
  }
}
```

下面对上述代码进行解释：BufferedReader，分行读入文件。逗点分割处理可以使用JDK 1.4 提供的 split 方法。另外，如果现在表中的列数小于读入数据的列数的话，少的部分可以作为列来追加。可以通过生成 TableItem 项目实现向表追加数据。使用 TableItem 项目的 setText 方法，设定读入值。最后，在 finally 语句中，关闭 BufferedReader。

（6）CSV 文件读入后的处理。接下来进行 CSV 文件读入后的相关处理。

```
private void loadEnd() {
mItemOpen.setEnabled(true);
long end = System.currentTimeMillis();
statusBar.setText("读入结束 : 处理時間"
        + (end - beginTime) + "微秒");
 }
```

通过上面的处理，就可以解决[打开]菜单无效的问题了。除此之外，状态栏中的读入处理花费时间以微秒为单位显示。

（7）终了处理。下面代码处理的是“终了”菜单运行时调用的终了处理。

```
private void close() {
```

```
        shell.close();
        shell.dispose();
    }
```

（8）选择行的显示处理。当选中表中某行时，需要在状态条中显示出该行的行号，下面的代码处理就是针对此项要求进行的。

```
private void setTableSelectedIndex() {
    int index = table.getSelectionIndex();
    statusBar.setText(index + "行目");
}
```

这样，就按照式样的要求完成了所有处理的编码。

15.2.3　Widget 的事件注册侦听器处理

大多数应用程序逻辑是作为对用户事件的响应来实现的。在创建屏幕和一些小窗口之后，应该将侦听器添加至已经创建的一些小窗口，并且当触发事件时，将执行侦听器代码。本节将介绍如何对 Widget 的事件注册调用逻辑的侦听器。

1．菜单注册

在 open 方法中做成 menu，然后对各菜单进行注册侦听的处理。

```
        // 打开菜单
mItemOpen.addSelectionListener(new SelectionAdapter(){
          public void widgetSelected(SelectionEvente){
          getOpenFileName();
          if (fileName != null) {
          // CSV 文件读入事前处理
          loadBegin();
          // CSV 文件读入处理
          loadFile();
          // CSV 文件读入事后处理
          loadEnd();
          }
       }
});
//关闭菜单
mItemExit.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
      close();
    }
});
```

2．追加选择表时的处理

在 createTable 方法内，同菜单一样，对于表进行侦听器注册。

```
table.addSelectionListener(new SelectionAdapter() {
       public void widgetSelected(SelectionEvent e) {
         // 选择表时，状态栏的行号表示
        setTableSelectedIndex();
       }
 });
```

完成了上面的编码，在试着运行一下程序。当然，事先需要准备一个 CSV 文件，正确的结果如图 15-10 所示。



图 15-10　运行程序后的 CSVViewer 窗口

看到上面的结果，就证明 CSVViewer 的功能就全部完成了。

但是，实际上这个应用程序中还存在一些问题：读入行数多的 CSV 文件时，可能会出现在文件读入的过程中，应用程序自动停止的现象。同时，读入状况显示不出来，无法进行窗口的移动和最小化。这个问题与下面要讲到的线程有关。

15.3　线程问题

前面提到的问题，事实上并不是应用程序停止了，而是在执行 loadFile 方法的过程中，无法调用描述 GUI 变更的处理，这样显示和窗口的状态变化等事件的处理速度就会明显变慢。

图 15-11 表示 UI 线程处理过程。在 main 方法内的循环中，虽然调用了 Display 的 ReadAndDispath 方法，但是事实上是通过调用这个方法来进行显示画面和事件处理的。在 SWT 中，把生成显示项目并在循环内执行方法的线程称为 UI 线程。如果该 UI 线程调用处理花费的时间过长的话，那么应用程序看上去就像停止了一样。

15.3.1　本机事件调度

在任何 GUI 应用程序下面，不管它的语言或用户界面工具箱是什么，OS 平台都会检测 GUI 事件，并将它们放在应用程序事件队列中。尽管在不同的 OS 平台上机制稍微有些不同，但是基础是相似的。当用户单击鼠标、输入字符或者对窗口的外观进行处理时，OS 将生成应用程序 GUI 事件（例如，鼠标单击、击键或窗口绘制事件）。它确定哪个窗口和应用程序应当接收哪个事件，并将该事件放置在应用程序的事件队列中。

任何窗口化的 GUI 应用程序的底层结构都是事件循环。应用程序进行初始化，然后启动循环，它只从队列中阅读 GUI 事件，并相应地作出反应。在处理其中一个事件时完成的任何工作必须快速地进行，以便让 GUI 系统可以响应用户。

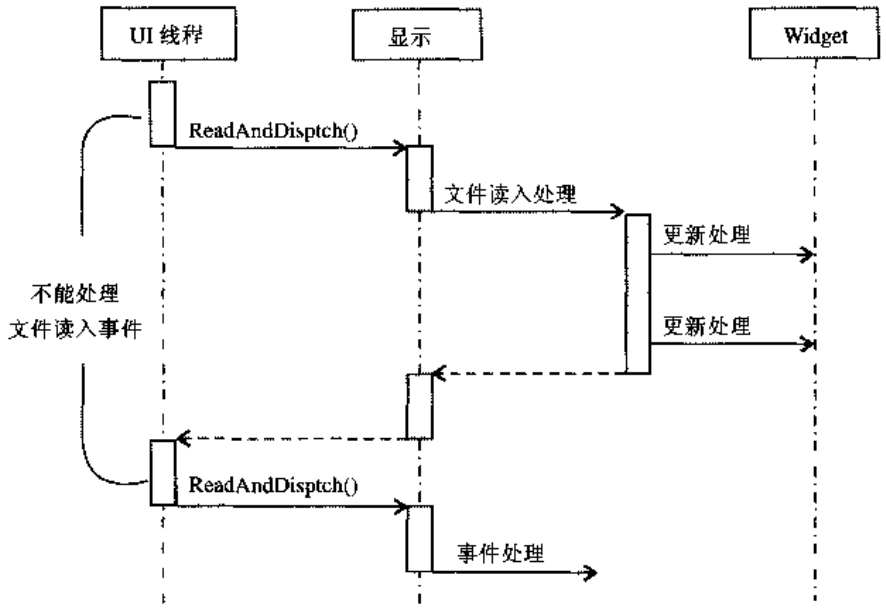


图 15-11　UI 线程处理过程

应该在单独的线程中执行由用户界面事件触发的长时间操作，以便允许事件循环线程快速返回，并从应用程序的队列中访存下一个事件。但是，必须利用显式锁定和序列化来控制从其他线程访问小窗口和平台 API。没有遵循规则的应用程序可能会导致 OS 调用失败，更糟糕的是，可能会锁定整个 GUI 系统。

15.3.2　工具箱用户界面线程

使用 C 语言的 GUI 程序员可能会熟悉使用平台事件循环的设计注意事项。但是，用 Java 编写的较高级别的小窗口工具箱通常试图通过隐藏平台事件循环来向应用程序开发者隐藏用户界面线程问题。

实现此过程的常见方法是设置专用工具箱用户界面线程，以便读取和调度事件循环，并将事件送至由正在单独线程中运行的应用程序服务的内部队列中。这允许工具箱有足够的时间来响应操作系统，而不对应用程序处理事件的时间设置任何限制。应用程序必须仍然使用特殊的锁定技术来从它们的应用程序线程中访问用户界面代码，它在整个代码中的完成应该是一致的，原因是所有应用程序代码正在非用户界面线程中运行。

尽管听起来好像可以“防止”应用程序发生用户界面线程问题，但是，实际上它将导致许多问题。当 GUI 事件的时间取决于 Java 线程实现和应用程序性能时，就很难调试和诊断问题。

目前的 GUI 平台对事件队列执行了许多优化。常见的优化就是将连续的绘制事件层叠到队列中。每当必须重新绘制窗口的一部分时，就可以检查队列是否存在绘制事件重叠或尚未调度的冗余绘制事件。可以将这些事件合并到一个绘制事件中，从而使得闪烁较少出现并且执行应用程序的绘制代码也不会太频繁。如果小窗口工具箱正在将事件快速拉出队列，并将它们发送到内部队列中，则此优化将会失败。

改变开发者对线程模型的理解会导致程序员在使用其他语言和工具箱来为本机 GUI 系统编程时发生混淆。

15.3.3 SWT 用户界面线程

SWT 遵循平台直接支持的线程模型。应用程序在它的主线程中运行事件循环，并直接从此线程中调度事件，这是应用程序的“UI 线程”。

注意：从技术上来说，UI 线程是创建显示的线程。实际上，此线程也是运行事件循环和创建小窗口的线程。

由于所有事件代码都是从应用程序的用户界面线程中触发的，因此，处理事件的应用程序代码可以自由地访问小窗口，并且不需要任何特殊技术就可以进行图形调用。然而，当响应某事件而执行长时间操作时，应用程序负责创建计算线程的分支。

注意：对于所有来自非用户界面线程的调用，而这些调用又必须来自用户界面线程，SWT 将触发 SWTException。

SWT 应用程序的主线程（包括事件循环）如下所示：

```
public static void main (String [] args) {
  Display display = new Display ();
  Shell shell = new Shell (display);
  shell.open ();
  while (!shell.isDisposed ()) {
     if (!display.readAndDispatch())
        display.sleep();
     }
     display.dispose ();
}
```

一旦创建了小窗口并且打开了 shell，应用程序就会读取和调度 OS 队列中的事件，直到关闭 shell 窗口为止。如果队列中没有事件可用，则说明显示进入睡眠状态，以使其它应用程序有机会运行。

注意：SWT 应用程序的最常见线程模型是在计算线程中运行单个用户界面线程并执行长时间操作。然而，SWT 不会限制开发者使用此模型。应用程序可以运行多个用户界面线程，而每个线程中都有单独的事件循环。

15.3.4 执行非用户界面线程中的代码

希望从非用户界面线程中调用用户界面的应用程序必须提供调用用户界面代码的可运行程序。显示类中的 syncExec（Runnable）和 asyncExec（Runnable）方法是用来在适当的时间在用户界面线程中执行这些可运行程序。

当非用户界面线程中的应用程序代码取决于用户界面代码的返回值时，应当使用

syncExec（Runnable），否则，需要确保可运行程序在返回到线程中之前能够完成运行。SWT 将在从应用程序的用户界面线程运行完可运行程序之前阻塞调用线程。例如，正在根据窗口的当前尺寸大小来计算某些内容的后台线程将要同步地运行代码以获取窗口的大小，然后继续它的计算。

当应用程序需要执行某些用户界面操作、但是与之前需要继续完成的操作无关时，应当使用 asyncExec（Runnable）。例如，更新进度指示器或者重绘窗口的后台线程可以请求异步更新，并继续它的处理。在此情况下，后台线程的时间与可运行程序的执行之间没有承诺关系。

15.3.5　线程化

在了解了线程的有关概念后，要想解决上面提到的问题，可以通过其他的数据流进行读文件的处理，如图 15-12 所示。

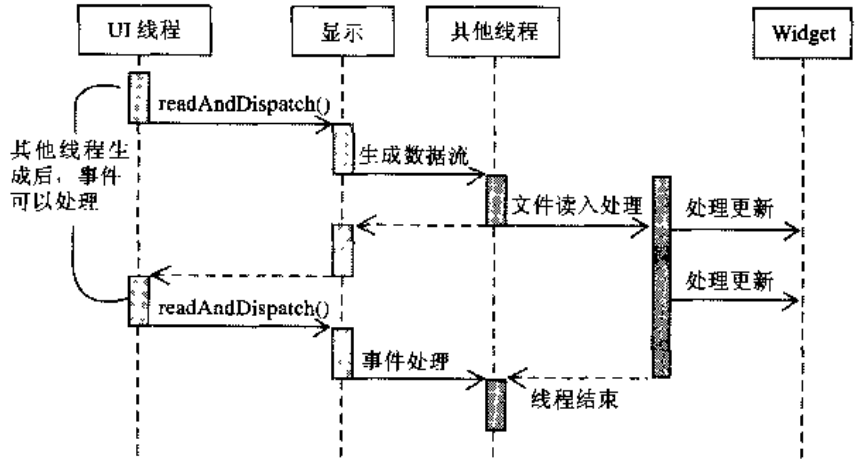


图 15-12　通过调用其他线程进行读文件的处理

具体步骤如下：

1．生成执行 Runnable 的类

为了能够在其他线程中进行读文件的处理，首先需要做一个执行 Runnable 的类。在读入处理中，CSVViewer 需要多次参照拥有实例变量的 Widget。当然可以制作一个不同于 CSVViewer 的源程序文件，但是为了方便起见，只在 CSVViewer 中制作一个内部类。

```
class CSVLoader implements Runnable {
    public void run() {
    }
}
```

2．修改[打开]菜单的处理

将调用 loadBegin，loadFile，loadEnd 的部分作一修改，目的是为了能在其他线程中执行。

```
// 打开菜单
mItemOpen.addSelectionListener(new SelectionAdapter() {
        public void widgetSelected(SelectionEvent e) {
        // 打开文件对话框，选择文件名
               getOpenFileName();
                   if (filename != null) {
                   // 其他线程执行
                           new Thread(new CSVLoader()).start();
                   }
        }
 });
```

3. 方法和区域的移动

把 loadBegin、loadFile、loadEnd 方法移动到 CSVLoader 中。另外，这些方法中使用到的实例变量 beginTine 也要移动到 CSVLoader 中。这样一来，在 run 方法内就可以调用这些方法了。

CSVLoader 部分的编码如下所示：

```
class CSVLoader implements Runnable {
   //用于测试的读入时间
   private long beginTime;
public void run() {
   // CSV 文件读入事前处理
  loadBegin();
  // CSV 文件读入处理
  loadFile();
  // CSV 文件读入事后处理
  loadEnd();
}
private void loadBegin() {
  // 打开菜单不可选择
 mItemOpen.setEnabled(false);
 ......
}
private void loadFile() {
  ......
}
private void loadEnd() {
  ......
  }
}
```

到此为止，就已经把读入处理移送到其他线程中去了。

4. 线程化的问题

如果将上述修改后的程序保存运行的话，会出现这样的结果：无法选择“打开”菜单，并且发生了如下的 SWTException 异常。

```
org.eclipse.swt.SWTException: Invalid thread access
at org.eclipse.swt.SWT.error(SWT.java:2330)
at org.eclipse.swt.SWT.error(SWT.java:2260)
at org.eclipse.swt.widgets.Widget.error(Widget.java:385)
at org.eclipse.swt.widgets.Widget.checkWidget(Widget.java:315)
at org.eclipse.swt.widgets.MenuItem.setEnabled(MenuItem.java:557)
at csvviewer.CSVViewer$CSVLoader.loadBegin(CSVViewer.java:173)
```

```
at csvviewer.CSVViewer$CSVLoader.run(CSVViewer.java:164)
at java.lang.Thread.run(Thread.java:534)
```

之所以会发生这样的异常，其原因在于操作 SWT 的 Widget 处理在前面提到的 UI 线程之外的某处被运行。在 SWT 中如果设定了这样的限制，对于同时具有多个线程的 GUI 操作，就可以简单而安全地进行了。在 SWT 中，遇到类似的有其他线程操作的情况时，可以使用 Display 类的 syncExec 与 asyncExec 方法，把这些处理移送到 UI 线程中具体过程可参照图 15-13 所示。

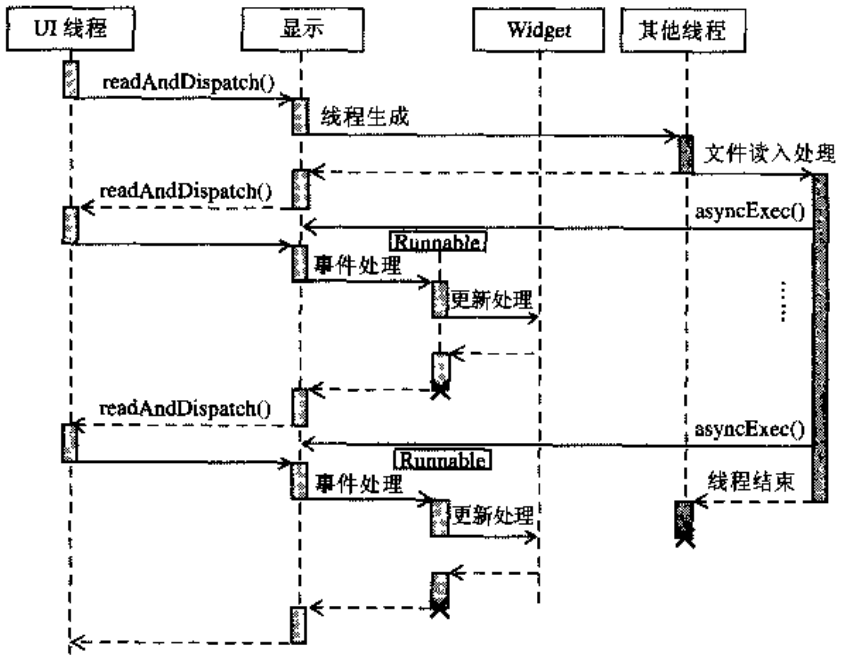


图 15-13　使用 asyncExec 方法将处理移送到 UI 线程中的过程

15.3.6　向 UI 线程移送处理

使用 Display 类的 asyncExec 方法将处理移送到 UI 线程中的过程如下。

1．追加 asyncExec 方法

虽说修改的对象是 loadBegin，loadFile，loadEnd 方法，但是首先需要追加 utility 方法。在多线程的环境下，可能会发生清除 display 的情况，比如，在处理过程中关闭窗口等，那么，需要在 CSVLoader 里追加下面的 utility 方法，代码如下：

```
private Boolean checkAsyncExec(Runnable r) {
        if (!display.isDisposed()) {
           display.asyncExec(r);
           return true;
        } else {
          return false;
        }
}
```

2. 修改 loadBegin 方法

把操作 SWT 的 Widget 的各个部分修改为在使用 checkAsyncExec 方法在 UI 流中运行。

```
private void loadBegin() {
    checkAsyncExec(new Runnable() {
            public void run() {

                    //[ 打开] 菜单设置为 "false" 状态
                mItemOpen.setEnabled(false);

                 // 文本框名的设定
                textBox.setText(fileName);

                 // 列数变化时表的再次生成
                Composite comp = table.getParent();
                table.dispose();
                createTable(comp);
                 comp.layout();

                // 状态栏的状态表示
                statusBar.setText("读入中...");
        }
 });

// 测试用读入时间
beginTime = System.currentTimeMillis();
}
```

3. 修改 loadFile 方法

与 loadBegin 类似，使用 checkAsyncExec 方法。在 while 循环中，如下修改：

```
private void loadFile() {
    ......
    while ((line = reader.readLine()) != null) {

    // 文字列分割
    final String[ ] datas = line.split(",");

    Boolean isOK = checkAsyncExec(new Runnable() {
        public void run() {
        if (table.isDisposed()) {
             return;
 }
 for (int I = table.getColumnCount(); I < datas.length; i++) {
  TableColumn column =
 new TableColumn(table, SWT.LEFT);
  column.setText("列" + (I + 1));
  column.setWidth(50);
 }
    // 行的设定
    TableItem item = new TableItem(table, SWT.NULL);
    item.setText(datas);
   }
});
if (!isOK) {
     return;
    }
  }
```

```
    ......
}
```

为了能够在 Runnable 的 run 方法内参照 loadFile 方法的 local 变量 datas，要将 datas 定义成 final 型。另外，当遇到 display 或者 table 一方被清除时，即使是处理继续进行也没有什么实际意义，所以可以把方法返回。

4. 修改 loadEnd 方法

跟 loadBegin 方法一样，使用 checkAsyncExec 方法。

```
private void loadEnd() {
    checkAsyncExec(new Runnable() {
      public void run() {
        if (mItemOpen.isDisposed() || statusBar.isDisposed()) {
          return;
        }
        mItemOpen.setEnabled(true);
        long end = System.currentTimeMillis();
        statusBar.setText(
          "读入完了 : 处理时间"
        +(end - beginTime) + "微秒");
    }
  });
}
```

在菜单或者状态栏被关闭的情况下，不需要继续进行处理所以可以将方法返回。

至此，CSVViewer 程序的运行速度问题已经解决了。

15.4　插件的用户界面框架（JFace）

JFace 是一个用户界面工具箱，它提供很难实现的、用于开发用户界面功能部件的 helper 类。JFace 在原始的小窗口系统的级别之上运行，使开发人员可以专注于实现特定插件的功能，而不必花费精力来处理底层小窗口系统，或者解决几乎在任何用户界面应用程序中都很常见的问题。JFace 提供如下四项用于处理常见的用户界面编程任务的类。

- 查看器：负责处理填充、排序、过滤和更新小窗口等最辛苦的工作。
- 操作和添加项：介绍用于定义用户操作的语义，并指定在何处提供它们。
- 图像和字体注册表：提供用于处理用户界面资源的常见模式。
- 对话框和向导：定义用于与用户进行复杂交互的框架。

SWT 和 JFace 之间的界线是很明显的。SWT 根本不依赖于任何 JFace 或平台代码。前几节所举的 SWT 示例说明了如何构建独立的应用程序。

JFace 用来在 SWT 库顶部提供常见的应用程序用户界面功能。JFace 并不试图“隐藏”SWT 或者替换它的功能。JFace 提供一些类和接口，以处理与使用 SWT 来对动态用户界面编程相关联的常见任务。通过考察查看器及它们与 SWT 小窗口之间的关系，就能更清楚地了解 JFace 和 SWT 之间的关系。

15.4.1 查看器

查看器允许在创建小窗口时使用模型对象。如果直接使用 SWT 小窗口，则需要将对象转换成 SWT 期望的字符串和图像。查看器充当 SWT 小窗口上的适配器，用于处理小窗口事件的常见代码，否则，将需要由用户自己来实现。

首先在 ReadmeSectionsView 的自述文件工具的视图添加项中实例化查看器。

```
public void createPartControl(Composite parent) {
        viewer = new ListViewer(parent);
        ...
}
```

注意：可以使用查看器来同时为工作台视图和编辑器提供实现。术语“查看器”并不意味着它们只能用于实现视图，例如，TextViewer 用于实现许多工作台和插件编辑器。

1. 标准查看器

JFace 为 SWT 中大多数重要的小窗口都提供了查看器。查看器通常用于列表、树、表和文本小窗口。

每个查看器都有相关联的 SWT 小窗口。可以通过在查看器构造函数中提供父组合体来隐式地创建此小窗口，或者通过先创建此小窗口，然后将此小窗口提供给它的构造函数中的查看器，从而显式地创建此小窗口。

有两种类型的查看器：

（1）面向列表的查看器。列表、树和表共享许多从用户的观点来说较为常见的一些功能，例如，填充对象、选择、排序和过滤。这些查看器保存域对象（称为元素）的列表，并将这些列表显示在其相应的 SWT 小窗口中。列表查看器知道如何从列表中的任何元素中获取文本标号。它从可以在查看器上设置的 ILabelProvider 中获取标号。列表查看器知道如何从小窗口映射回到查看器客户所知道的元素中。

使用平面 SWT 小窗口的客户需要在这样一个 SWT 级别上运行——在该级别中，项是字符串，而事件通常与字符串列表中的索引相关。查看器提供了更高级的语义。对于客户，是面对使用这些 SWT 小窗口提供给查看器的元素来通知对列表的选择和更改的。查看器处理下列各项事务中的所有繁琐工作：将索引映射回至元素、调整对象的过滤视图以及在必要时重新排序。

过滤和排序功能是通过为查看器指定查看器排序器（ViewerSorter）或查看器过滤器（ViewerFilter）来处理的。（除了列表查看器之外，还可以为树查看器和表查看器指定这些。）客户只需要提供可以比较或过滤列表中的对象的类。查看器根据指定次序和过滤器来处理填充列表的详细信息，并在添加和除去元素时维护次序和过滤器。

（2）文本查看器。文本小窗口具有许多常见语义，例如，双击行为、撤销、着色以及按索引或者行来导航。TextViewer 是 SWT StyledText 小窗口的适配器。文本查看器为客户提供了文档模型，并管理从文档到由文本小窗口提供的样式文本信息的转换。

2. 查看器体系结构

要了解查看器，必须熟悉查看器的输入元素，了解输入无素的内容和选项，以及在它正在处理的小窗口中实际显示的信息之间的关系。具体相关内容如下：

（1）输入元素。输入元素是查看器正在显示（或编辑）的主要对象。从查看器的角度来看，输入元素可以是任何对象。不建议由输入元素来实现任何特定接口。查看器必须能够处理输入元素的更改。如果在查看器中设置了新的输入元素，则它必须根据新元素来重新填充它的小窗口，并取消它自己与前一输入元素的关联。对于每种类型的查看器，注册为输入元素上的侦听器的语义和根据输入元素填充小窗口的语义是不同的。

（2）内容查看器。内容查看器是一种具有严格定义的协议，用来获取来自于它的输入元素的信息。内容查看器使用两个专门的 helper 类——IContentProvider 和 ILabelProvider 来填充它们的小窗口并显示关于输入元素的信息。

IContentProvider（内容提供程序）提供了基本的有效期协议，用以将内容提供程序与输入元素关联起来以及处理输入元素的更改。IContentProvider 为不同类型的查看器实现了多个专门的内容提供程序。最常用的内容提供程序是 IStructuredContentProvider，它可以提供给定输入元素的对象列表，用于类似列表的查看器，例如，列表、表或树。通常，内容提供程序知道如何在输入元素和期望的查看器内容之间进行映射。

ILabelProvider（标号提供程序）更加深入，它给定查看器的内容，是从输入元素和内容提供程序中派生出来的。ILabelProvider 可以生成在查看器中显示内容所需的特定用户界面元素，例如，名称和图标。标号提供程序可以帮助保存图标资源，因为它们可以确保用于查看器中所有相似类型图标的相同实例的图标都具有相同的实例。

注意：特定的内容和标号提供程序的实例并不会在多个查看器之间共享。即使所有查看器都使用相同类型的内容或标号提供程序，也应该利用每个查看器自己的内容提供程序的实例来进行初始化。内容提供程序有效期协议用于内容提供程序与它的查看器之间“一对一”的关系。

输入元素、内容提供程序和标号提供程序允许查看器隐藏有关填充小窗口实现的大部分详细信息。查看器的客户只需要关心是否使用了正确类型的输入和内容提供程序来填充查看器。标号提供程序必须知道如何从查看器内容中派生出用户界面信息。

15.4.2　操作和添加项

操作类允许定义用户命令而不考虑它们在用户界面中的表示。这使得能够灵活地更改操作在插件中的表示，不必在选择某个命令时更改实际执行该命令的代码。添加项用来管理表示命令的实际用户界面项。用户不需要对添加项编程，但是却能在某些工作台和 JFace API 中看到它们。

下面分别详细介绍操作、添加项及添加项管理器。

1. 操作（IAction）

操作表示由最终用户触发的命令。操作通常与按钮、菜单项和工具栏中的项相关联。尽管操作本身不放置在用户界面中，但是它们确实具有面向用户界面的属性，例如，工具提示文本、标号文本和图像。这允许其他类为操作的表示构造小窗口。

当用户触发用户界面中的操作时，就会调用该操作的运行方法来执行实际工作。运行方法中的常见模式是查询工作台，选择并处理所选的对象。另一种常见模式是在选择操作时启动向导或对话框。

用户不必直接实现 IAction 接口，而是应该在子类中实例化操作类。用户可以浏览此类的子类，以便了解操作的许多常见模式。以下代码实现了“关于”操作，它是工作台中最简单的操作之一。

```
public void run() {
   new AboutDialog(workbenchWindow.getShell()).open();
}
```

工作台操作代理是利用与它们相关联的视图或编辑器来初始化的，工作台操作代理可以浏览工作台页面或窗口，访问执行操作所需要的选择或任何其他信息。当想要在插件中定义命令时，就将实现自己的操作类。如果是向其他视图和编辑器添加操作，则将实现操作代理。

2. 添加项（IContributionItem）

添加项表示操作的用户界面部分。更准确地说，它表示添加到共享用户界面资源的项，例如，菜单或工具栏。添加项知道如何利用表示该添加的相应 SWT 项来填充特定的 SWT 小窗口。如果想为工作台用户界面添加操作，则不必费心再创建添加项。当工作台为已定义的操作创建用户界面项时，已经以用户的名义完成了此任务。

3. 添加项管理器（IContributionManager）

添加项管理器表示将在用户界面中提供的添加项的集合。可以通过使用已命名的添加标识来添加和插入添加项，以便按照相应的次序来放置这些添加项。还可以根据标识来查找添加项和除去个别项。

IContributionManager 的每种实现都知道如何使用 SWT 小窗口的项来填充该小窗口。JFace 为菜单（IMenuManager）、工具栏（IToolBarManager）和状态行（IStatusLineManager）提供了添加项管理器。

作为插件开发者不需要实现这些接口，但在 API 方法中，却能看到对其中某些管理器的引用。

15.4.3 用户界面资源

org.eclipse.jface.resource 包定义用来帮助插件管理用户界面资源（例如，字体和图标）的类。

许多工作台扩展点允许插件提供可以用来显示它们在工作台中的添加项的图标。由

于 GUI 操作系统的内存只能同时支持有限数量的图像或字体，因此，必须小心管理插件的用户界面资源，有时在小窗口之间还要共享这些资源。

JFace 提供了一些基本支持类，它们允许插件管理自己的图标和字体，而不用担心何时创建和销毁相应的平台图形对象。这些支持类可以直接由插件来使用，也可以在工作台使用这些类来获取在扩展点标记中所描述的图像时间接使用。

下面具体介绍一下 JFace 都提供了哪些基本支持类。

1. 图像描述符和注册表

SWT 图像类表示来自操作系统的透视图的图像。因为大多数 GUI 操作系统都对可同时打开的图像数量有限制，所以，在创建 SWT 图像类时应当特别小心，并且确保在使用完它们时，还要对它们进行正确地销毁。通过使用 JFace 的 ImageDescriptor 和 ImageRegistry 类而不是 SWT 图像，插件通常都可以避免直接创建、管理和处理这些图像。

下面分别来介绍图像描述符（Image Descniptor 类）和图像注册表（I_类）。

（1）图像描述符。ImageDescriptor 类可以用作图像的轻量级描述。它指定创建图像所需的所有内容，例如，可以从其中获得图像的 URL 或文件名。ImageDescriptors 不会分配实际的平台图像，除非使用 createImage() 方法来特别请求。

如果代码构造为在一个位置定义所有图标，并在需要它们时进行分配，则图像描述符是最佳选择。用户可以在任何时候创建图像描述符，而不用担心操作系统资源，全部在初始化代码中创建它们是很方便的。

（2）图像注册表。ImageRegistry 类用来保存已命名的图像的列表。用户可以直接将图像描述符或 SWT 图像添加到列表中。在注册表中按名称来请求图像时，注册表将返回该图像（如果创建了该图像），或者根据描述符来创建一个图像（如果未创建的话）。允许注册表的用户共享图像。

因为图像是由多个用户共享的，任何用户不能销毁添加到注册表中的图像或者从注册表中检索到的图像。而是由注册表来负责处理图像。当平台 GUI 系统关闭时，注册表就自动销毁图像。

2. 使用图像的插件模式

（1）在 plugin.xml 中指定图像。可能的话，在 plugin.xml 文件中指定插件的用户界面对象的图标。许多工作台扩展点都包含用于图标文件的配置参数。通过在 plugin.xml 的扩展添加项中定义图标，这就使图像管理模式保留在了平台上。由于通常将图标保存在插件的目录中，因此允许在同一个位置指定图标和管理文件。只有当不能将图标指定为扩展添加项的一部分时，才应考虑使用其他模式。

（2）显式创建。当图像不是频繁使用并且也不共享时，显式地创建图像是最好方式。可以直接在 SWT 中创建图像，并在使用完图像之后进行销毁，也可以通过使用 ImageDescriptor 并调用 createImage() 方法来显式地创建图像。在第一种情况下，当使用完图像之后，必须调用图像的 dispose() 方法来销毁图像。例如，如果在对话框打开的情况下创建了图像，则在关闭对话框时就应该销毁该图像。

（3）图像注册表。当在插件中频繁地使用图像并且在用户界面中的许多不同的对象之间共享图像时，向 ImageRegistry 注册图像描述符是很有用的。注册表中的图像将与按同一名称来查询图像的任何对象共享。一定不能销毁注册表中的任何图像，因为其他对象也共享这些图像。

当图像可能在插件的整个有效期内频繁使用，并且被许多对象共享时，将图像添加到图像注册表中是最好的方式。使用注册表的缺点是，在关闭 GUI 系统之前不会销毁注册表中的图像。由于对可同时打开的平台（SWT）图像数量有限制，因此，不要在注册表中注册太多的图标。

AbstractUIPlugin 类中包含关于创建插件宽幅图像注册表的协议。

（4）标号提供程序。当频繁使用图标来显示特定查看器中的项时，可以通过使用标号提供程序在查看器中的相似项目之间共享这些图标。由于标号提供程序负责为查看器中的任何对象返回图像，因此，它可以控制图像的创建以及在查看器中的对象之间共享图像。

标号提供程序可以使用先前讨论的任何技术来生成图像。如果浏览 LabelProvider 子类中的 getImage()的各种实现，则用户将看到多种方法，包括高速缓存对象的单个图标和按类型来维护图像的表。必须在标号提供程序的 dispose()方法中销毁由标号提供程序创建的图像，该方法是在销毁查看器时调用的。

使用标号提供程序是在显式创建和图像注册表之间的一种很好的折衷。它促进了类似图像注册表之类的图标的共享，但是仍然保持对实际图像的创建和处理进行控制。

（5）插件的图像类。当精确调整插件时，通常都会对所有这些不同的图像创建模式进行试验。将有关图像创建的决策制定隔离在一个单独的类中，即插件的图像类，并指导所有客户使用该类来获取所有图像，这样就可以调整创建顺序，以便反映插件的实际性能特征。

3. 字体注册表

字体是平台操作系统中另一个受限制的资源。字体的创建和处理问题与图像是相同的，需要类似的速度-空间折衷。通常，在 SWT 中，字体的分配是通过使用平台的相关字体名来请求字体来实现的。

FontRegistry 类按照字体的名称来保存字体表。它管理字体的分配和处理。通常，插件应当避免分配任何字体或描述具有特定于平台的名称的字体。尽管字体注册表是在 JFace 内部使用的，但是插件通常不使用字体注册表。JFaceResources 类应当用来访问公共字体。

允许用户在首选项页面中指定应用程序的首选字体是很常见的。在这些情况下，应该使用 FontFieldEditor 来获取用户的字体名，并使用 FontRegistry 来保存字体。FontFieldEditor 仅在首选项页面中使用。

JFaceResources 类控制对常见平台字体和图像的访问。它维护内部字体和图像注册表，以便客户可以共享已命名的字体和图像。在工作台和其他插件中使用了很多技术以便在需要的地方共享图像。JFaceResources 图像注册表未在工作台和插件代码中广泛使用。

字体的使用更简单——工作台和大多数插件使用 JFaceResources 类来根据逻辑名请求字体，提供了诸如 getDialogFont()和 getDefaultFont()到方法，以便插件可以在它们的用户界面中使用期望的字体。

15.4.4　长时间运行的操作

org.eclipse.jface.operations 包为需要进度指示器或允许用户取消操作等长时间运行的操作定义接口。

1．可运行程序和进度

平台核心运行时定义常见接口 IProgressMonitor，在运行长时间运行的操作时，它用来向用户报告进度。在为用户显示进度很重要的情况下，客户可以提供监视器来作为许多平台 API 方法中的参数。

JFace 为实现进度监视器的用户界面的对象定义了多个特定接口。具体可参照 Eclipse 帮助文档。

2．模态操作

提供 ModalContext 类用来运行客户代码的透视图是模态的操作。它用于 IRunnableContext 的不同实现。如果插件需要等待长时间运行的操作完成才能继续执行，则可以使用 ModalContext 来完成此任务，并且仍然让用户界面保持响应。

在模态上下文中运行操作时，可以选择将该操作分配到不同的线程中。如果派生指令为 false，则操作将在调用的线程中运行。如果派生指令为 true，则操作将在新线程中运行，调用线程将停止，并且运行用户界面事件循环，直到该操作终止为止。

本章小结

本章主要介绍了 SWT 以及使用 Eclipse 制作 SWT 应用程序的方法，另外还向读者介绍了在 SWT 应用程序中使用 UI 流之外的其他流的方法，使用 asyncExec 方法进行简单的调整的方法等。在最后一节中，还简单介绍了 JFace 有关方面的知识。通过本章的学习，相信读者能够使用 SWT 结合 GUI 的组件，非常方便地制作 SWT 的应用程序。

附录 A　部分专业术语中英文对照

英文缩写	英 文 全 称	中 文 术 语
API	Application Programming Interface	应用程序接口
ASP	Active Server Pages	活动服务器页面
AWT	Abstract Window Toolkit	抽象窗口工具集
BMP	Bean-managed Persistence	Bean 管理的持久实体 Bean
CDT	C/C++ Development Tools	C 语言开发工具
CMP	Container-managed Persistence	容器管理的持久实体 Bean
CPL	Common Public License	通用公共许可证
CVS	Concurrent Versions System	并发版本控制系统
DBMS	Date Base Mananement System	数据库管理系统
DDL	Data Definition Language	数据定义语言
DLL	Dynamic Link Library	动态连接库
DTD	Document Type Definition	文件类型定义
EJB	Enterprise JavaBeans	企业级 JavaBeans
GUI	graphics user interface	图形用户接口
I/O	Input/Output	输入/输出
IDE	Integrated Development Environment	集成开发环境
ISV	Independent Software Vendor	独立软件提供商
J2EE	Java 2 Platform, Enterprise Edition	Java 平台企业版
JDBC	Java Database Connectivity	Java 数据库连接
JDK	Java Development Kit	Java 开发工具包
JDT	Java Development Tools	Java 开发工具
JIT	Just-In-Time	即时
JMX	Java Manage eXtension	Java 管理扩展
JNI	Java Native Interface	Java 本地接口
JRE	Java Run Environment	Java 运行时环境
JVM	Java Virtual Machine	Java 虚拟机
LDAP	Lightweight Directory Access Protocol	轻量目录访问协议
MinGW	Minimalist GNU for Windows	以 GNU 为基础的并发 C/C++项目的工具箱

（续表）

英文缩写	英 文 全 称	中 文 术 语
MVC	Model-View-Controller	模型-视图-控制器
NLS	National Language Support	区域语言支持
OS	Operating System	操作系统
PHP	Hypertext Preprocessor	超文本预处理器
RCS	Revision Control System	修改控制系统
SCCS	Source Code Control System	源码控制系统
SDK	Software Development Kit	软件开发工具包
SWT	Standard Widget Toolkit	标准小窗口工具箱
UI	User Interface	用户接口
UML	Unified Modeling Language	统一建模语言
URI	Uniform Resource Identifier	统一资源标识符
URL	Uniform Resource Locator	统一资源定位符
WMF	Windows Meta File	Windows 图元文件
XMI	XML Metadata Interchange	XML 源数据交换
XML	Extensible Markup Language	可扩展标记语言
XP	eXtreme Programming	极限编程

附录 B　部分资源下载地址

资源名称	版　本	下载网址
Apache 服务器	Apache 2.0.50	http://httpd.apache.org/
C/C++编译器	MinGW	http://sourceforge.net/projects/mingw/
C/C++开发工具	CDT 2.0	http://www.eclipse.org/cdt/
Eclipse	Eclipse 3.0	http://www.eclipse.org/downloads/index.php
Java 软件开发工具箱	J2SDK 1.4.2	http://java.sun.com/products/archive/
JBoss 服务器	JBoss 4.0	http://www.jboss.org/downloads/index
Lomboz 插件	Lomboz (Eclipse 3.x stream)	http://www.objectlearn.com/
MySQL	MySQL 5.0	http://www.mysql.com/downloads/index.html
Struts 插件	EasyStruts 1.0.1	http://easystruts.sourceforge.net/
Struts 框架	Struts 1.1	http://jakarta.apache.org/site/binindex.cgi
Tanghan 插件	TanghanPlugin_0.1.0.3m5	http://forum.hibernate.org.cn/download/jiinux/
Tomcat 插件	TomcatPlugin 3.0	http://www.sysdeo.com/eclipse/tomcatPlugin.html
Tomcat 服务器	Tomcat 5.0.27	http://jakarta.apache.org/site/binindex.cgi
UML 插件	EclipseUML 1.2.1	http://www.eclipseuml.com/download/index.jsp
多国语言包插件	2.1.2_Translations	http://download2.eclipse.org/downloads/